TEXAS INSTRUMENTS

99000

TMS99105A and TMS99110A 16-Bit Microprocessors

MICROPROCESSOR SERIES™

Preliminary Data Manual

ADVANCE INFORMATION

This document contains information on a
new product. Specifications are subject
to change without notice.

NOTICE

Texas Instruments reserves the right to make changes at any time in order to improve design and to supply the best product possible. Information contained in this publication is believed to be accurate and reliable. However, responsibility is assumed neither for its use nor for any infringement of patents or rights of others that may result from its use. No license is granted by implication or otherwise under any patent or patent right of Texas Instruments or others.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS, COMMANDS AND CODES

REFERENCE	DEFINITION
ABS	Absolute value
A/D (bus)	Address data bus
ALATCH	Address latch
AP	Attached processor
APP	Attached processor present
AUMS	Arithmetic logic unit, Macrostore access MPILCK inactive
AUMSL	Arithmetic logic unit, Macrostore access MPILCK asserted
BLWP	Branch and load workspace pointer
BST	Bus status code
CLKOUT	Clock output signal
CRU	Communications register unit
DMA	Direct memory access
DOP (bus code)	Destination operand transfer
EIST	Error interrupt status
EVAD	Evaluate address instruction
GM (bus code)	General memory transfer
HOLD	Self-explanatory
HOLDA (bus code)	Hold acknowledge
IAQ (bus code)	Instruction acquisition
ILLOP	Illegal operation
INTA (bus code)	Interrupt acknowledge
INTREQ	Interrupt request
IO (bus code)	I/O transfer
IOP (bus code)	Immediate data, symbolic address
LDCR	Load CRU, output instruction
LDD	Long distance destination
LDS	Long distance source
LST	Load status
LSW	Least-significant word
MEM	Memory cycle
MID(bus code)	Macroinstruction detected
MPILCK	Multiprocessor interlock
MSW	Most-significant word
NMI	Non-maskable interrupt
PC	Program counter
PSEL	Memory page select
RD	Read enable
RESET (bus code)	Reset. RESET input is pulled low
RTWP	Return from subroutine or interrupt
R/W	Read/Write
SBO	Set bit to one
SBZ	Set bit to zero
SOP (bus code)	Source operand transfer, MPILCK inactive
SOPL (bus code)	Source operand transfer, MPILCK asserted
ST (bus code)	Status register update
STCR	Store CRU input instruction
TB	Test bit
TCMB	Test-and-clear memory bit
TSMB	Test-and-set memory bit
WS	Workspace
WE/IOCLK	Write enable and inverted I/O clock
WP	Workspace pointer
XOP	Extended operation

1. INTRODUCTION

1.1 DESCRIPTION

The TMS99000 series is a third generation family of single-chip 16-bit micrroprocessors and advanced peripherals, using N-channel silicon-gate SMOS technology. The TMS99000 family of processors offers unprecedented speed and a powerful instruction set that is an opcode-compatible enrichment of the TMS9900 and TMS9995 instruction set. These processors build on the unique memory-to-memory architecture that was pioneered at Texas Instruments and feature multiple register files, resident in memory, to permit faster response to interrupts and increased programming flexibility.

The TMS99000 family includes two microprocessors, the TMS99105A and the TMS99110A, which are identical except for specialized programmations of the on-chip Macrostore memory. The ROM macrostore in the TMS99110A microprocessor contains floating point instructions as part of the machine language instruction set. The TMS99105A microprocessor contains RAM macrostore while the TMS99110A contains both RAM and ROM macrostore.

Texas Instruments manufactures a complete set of MOS and TTL integrated circuits to provide memory and logic functions for the TMS99000 system. The system is fully supported by software and a complete prototyping system.

All references in this document, unless explicitly indicated, refer to all members of the TMS99000 family of microprocessors.

1.2 KEY FEATURES

- 16-bit instruction word
- Memory-to-memory architecture
- Instantaneous access to 256K bytes of memory
- 84-instruction superset of TMS9900 instruction set
  - SIGNED multiply and divide
  - Long-word (32-bit) shift, add, subtract
  - Load status register, load workspace pointer
  - Stack support – branch and push link, branch indirect
  - Multiprocessor support – test, test and clear, test and set
- Privileged mode
- Macrostore* emulation of user-defined instructions
- Status signals to identify processor activity
- Interrupt acknowledge signal
- Arithmetic fault interrupt
- Illegal instruction interrupt
- 16 prioritized hardware interrupts
- 16 software interrupts (XOPS)
- Programmed I/O
- DMA compatible
- Bit- , byte- and word-addressable I/O
- Multiprocessor system interlock signal (hardware support for indivisible operations on semaphores)
- Attached processor interface
- N-channel silicon-gate SMOS technology
- 167 nsec machine cycle time
- On-chip clock generator and oscillator

*Macrostore is a trademark of Texas Instruments Incorporated

- 40-pin package
- Single +5 volt supply

2. ARCHITECTURE

2.1 MEMORY ALLOCATION

The memory word of the processor is 16 bits long as shown in Figure 1. Words are assigned even-numbered addresses in memory. The contents of each memory word can also be treated as two bytes of eight bits each. The instruction set supports both word and byte operations. A 16-bit address is explicitly manipulated by all memory addressing modes, but only the 15-bit word address is provided to the memory system. This allows direct addressing of 64K bytes of memory space, referred to as the logical address space. The instantaneous address reach of the processor may be increased to 256K bytes using the techniques described in Section 3.2.3.

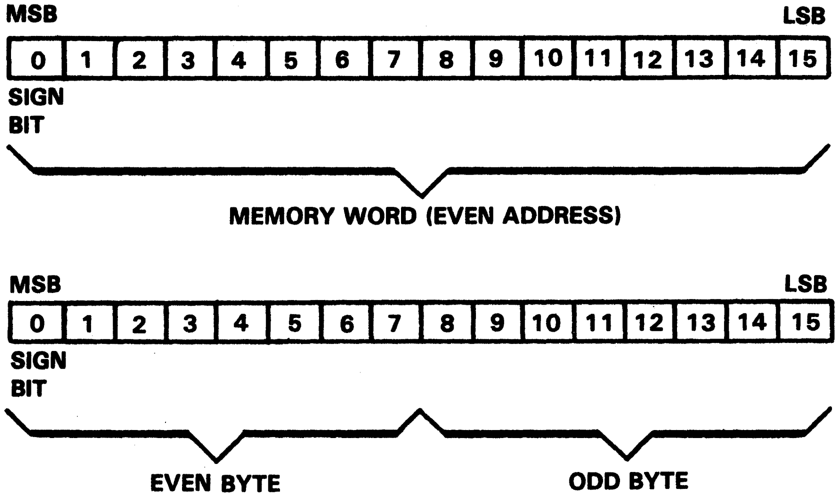
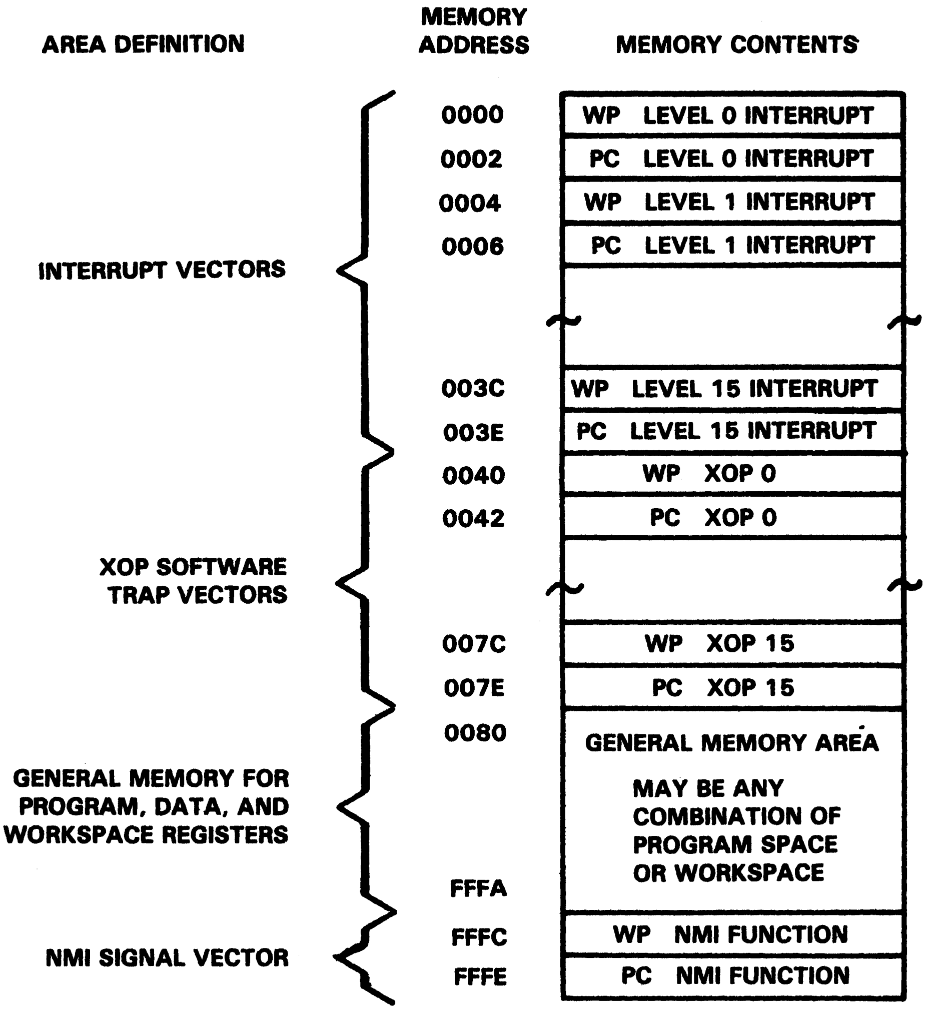


FIGURE 1 – WORD AND BYTE FORMATS

Byte instructions may access either byte as necessary. Byte instructions involving workspace register data operate on the most-significant byte (even address in Figure 1) of the workspace register, and leave the least-significant byte (odd address) unchanged. The two bytes in a register can be swapped using the SWPB instruction. Additionally, since the workspace resides in memory, the least-significant byte of a register may be addressed, if desired, using any of the general memory addressing modes.

The processor memory map (Figure 2) shows the locations in the memory address space for the interrupt and XOP trap vectors, and the non-maskable interrupt (NMI). All remaining memory space is available for programs, data, and workspaces.



AREA DEFINITION	MEMORY ADDRESS	MEMORY CONTENTS
INTERRUPT VECTORS	0000	WP LEVEL 0 INTERRUPT
	0002	PC LEVEL 0 INTERRUPT
	0004	WP LEVEL 1 INTERRUPT
	0006	PC LEVEL 1 INTERRUPT
	003C	WP LEVEL 15 INTERRUPT
	003E	PC LEVEL 15 INTERRUPT
XOP SOFTWARE TRAP VECTORS	0040	WP XOP 0
	0042	PC XOP 0
	007C	WP XOP 15
	007E	PC XOP 15
GENERAL MEMORY FOR PROGRAM, DATA, AND WORKSPACE REGISTERS	0080	GENERAL MEMORY AREA MAY BE ANY COMBINATION OF PROGRAM SPACE OR WORKSPACE
	FFFA	
NMI SIGNAL VECTOR	FFFC	WP NMI FUNCTION
	FFFE	PC NMI FUNCTION

FIGURE 2 – MAP OF MAIN MEMORY ADDRESS SPACE

2.2 TMS99000 BLOCK DIAGRAM AND FLOW CHART

The block diagram of the processor is shown in Figure 3. A flowchart, representative of the processor functional operation, is shown in Figure 4.

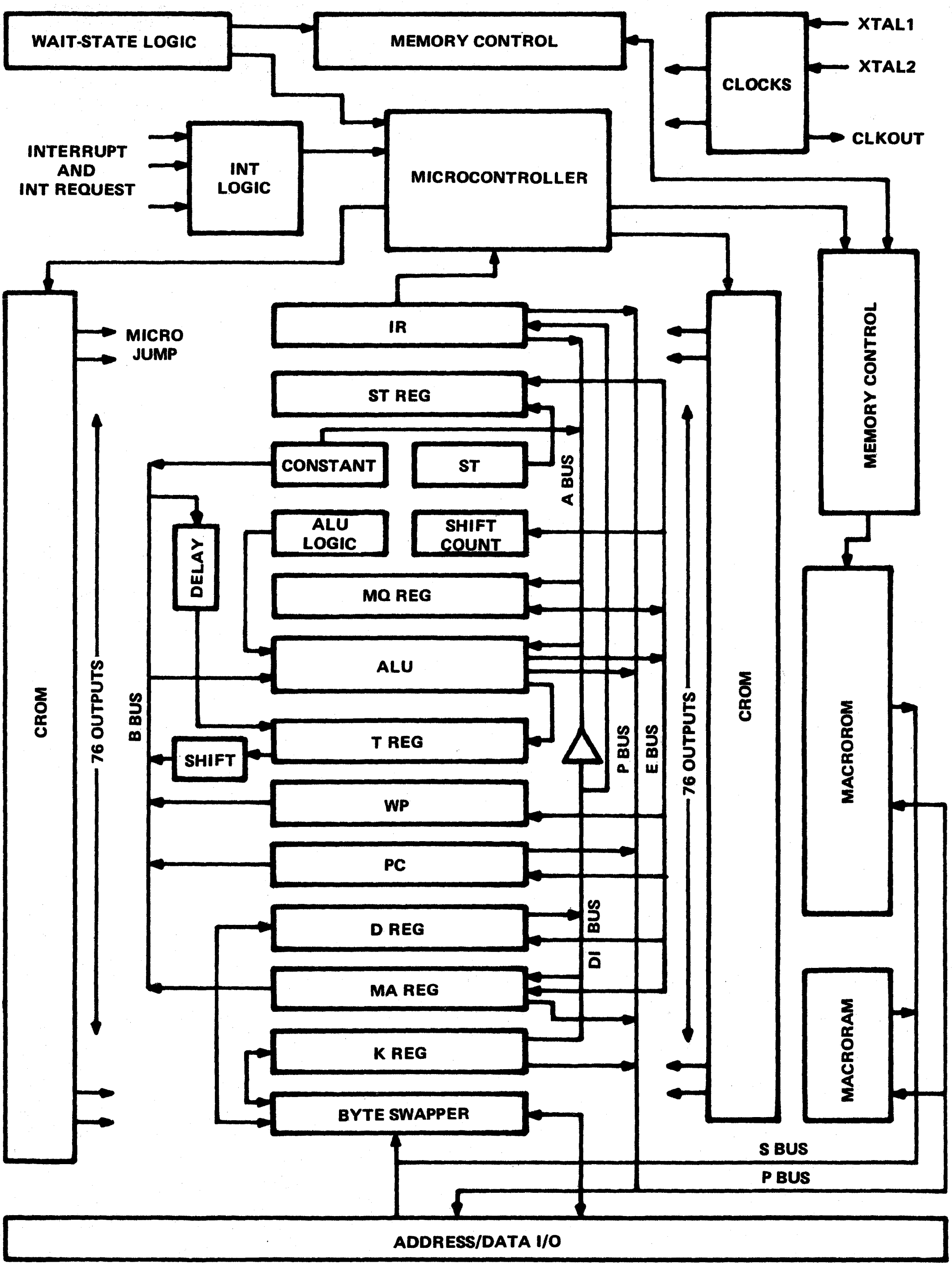
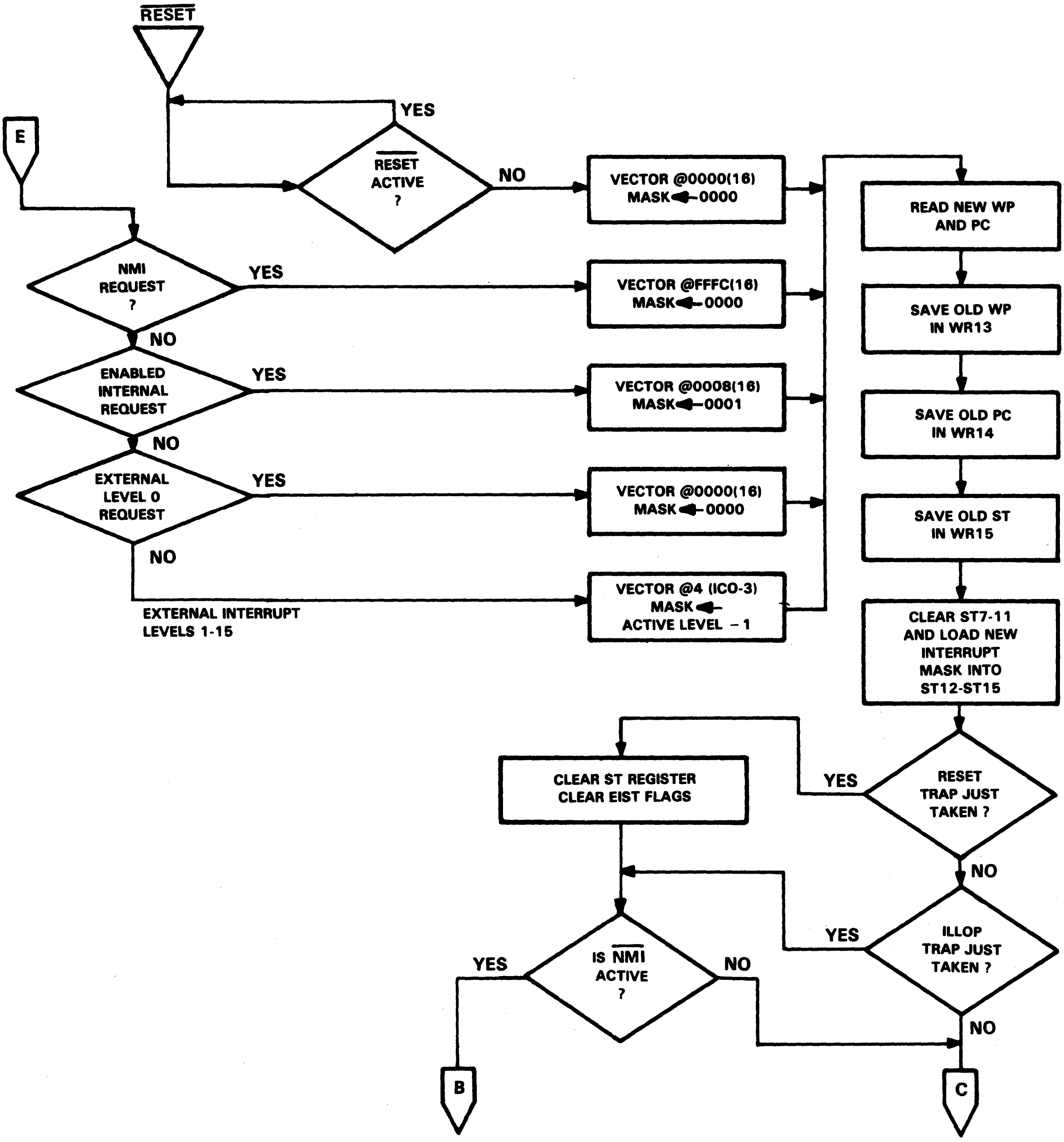


FIGURE 3 – BLOCK DIAGRAM OF TMS99000



$\overline{\text{RESET}}$
E
YES
$\overline{\text{RESET}}$ ACTIVE ?
NO
VECTOR @0000(16) MASK◄—0000
READ NEW WP AND PC
NMI REQUEST ?
YES
VECTOR @FFFC(16) MASK◄—0000
SAVE OLD WP IN WR13
NO
ENABLED INTERNAL REQUEST
YES
VECTOR @0008(16) MASK◄—0001
SAVE OLD PC IN WR14
NO
EXTERNAL LEVEL 0 REQUEST
YES
VECTOR @0000(16) MASK◄—0000
SAVE OLD ST IN WR15
NO
EXTERNAL INTERRUPT LEVELS 1-15
VECTOR @4 (ICO-3) MASK◄— ACTIVE LEVEL – 1
CLEAR ST7-11 AND LOAD NEW INTERRUPT MASK INTO ST12-ST15
CLEAR ST REGISTER CLEAR EIST FLAGS
YES
RESET TRAP JUST TAKEN ?
NO
YES
ILLOP TRAP JUST TAKEN ?
YES
IS $\overline{\text{NMI}}$ ACTIVE ?
NO
NO
B
C

FIGURE 4 – TMS99000 FLOWCHART

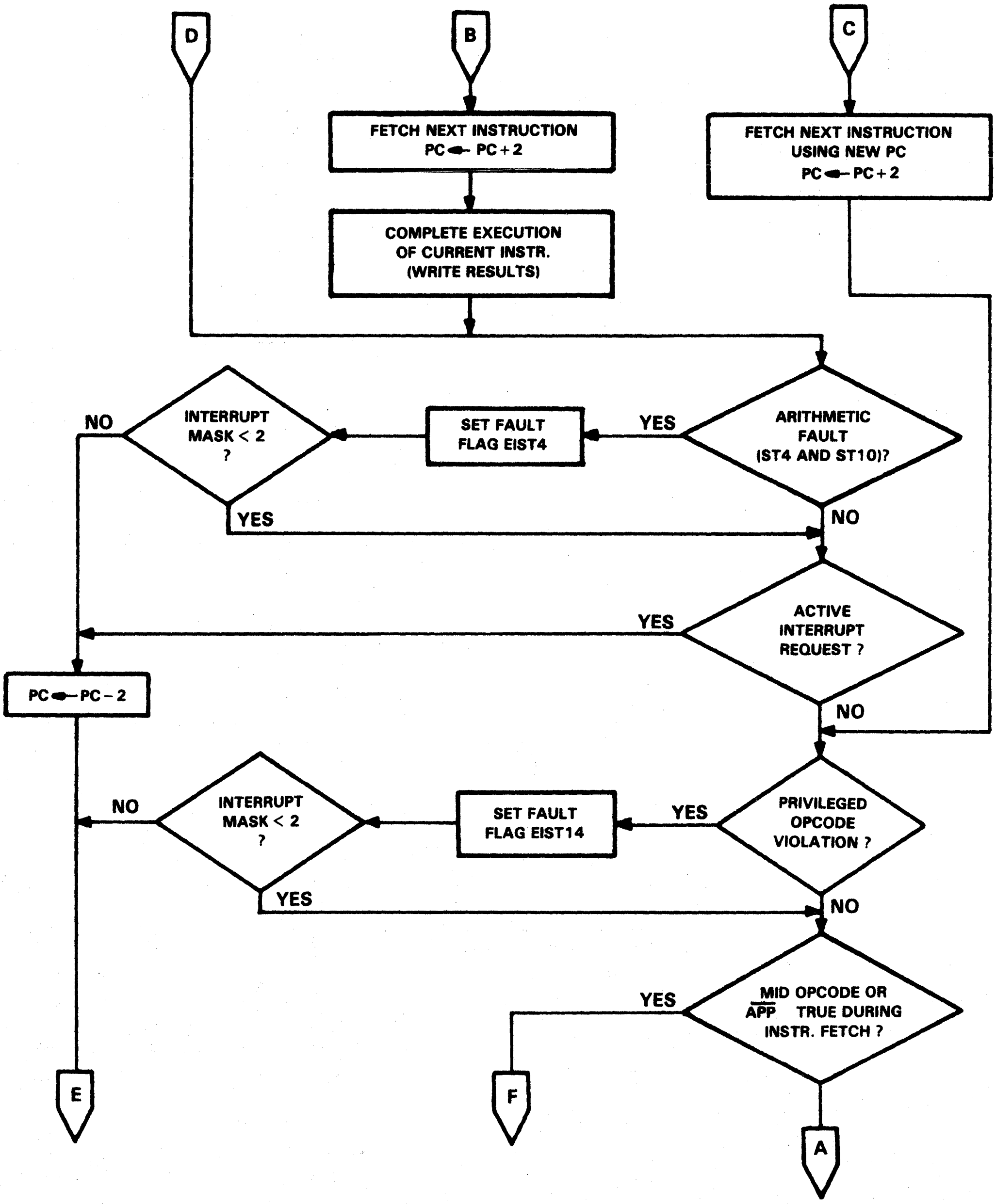


FIGURE 4 – TMS99000 FLOWCHART (CONT'D)

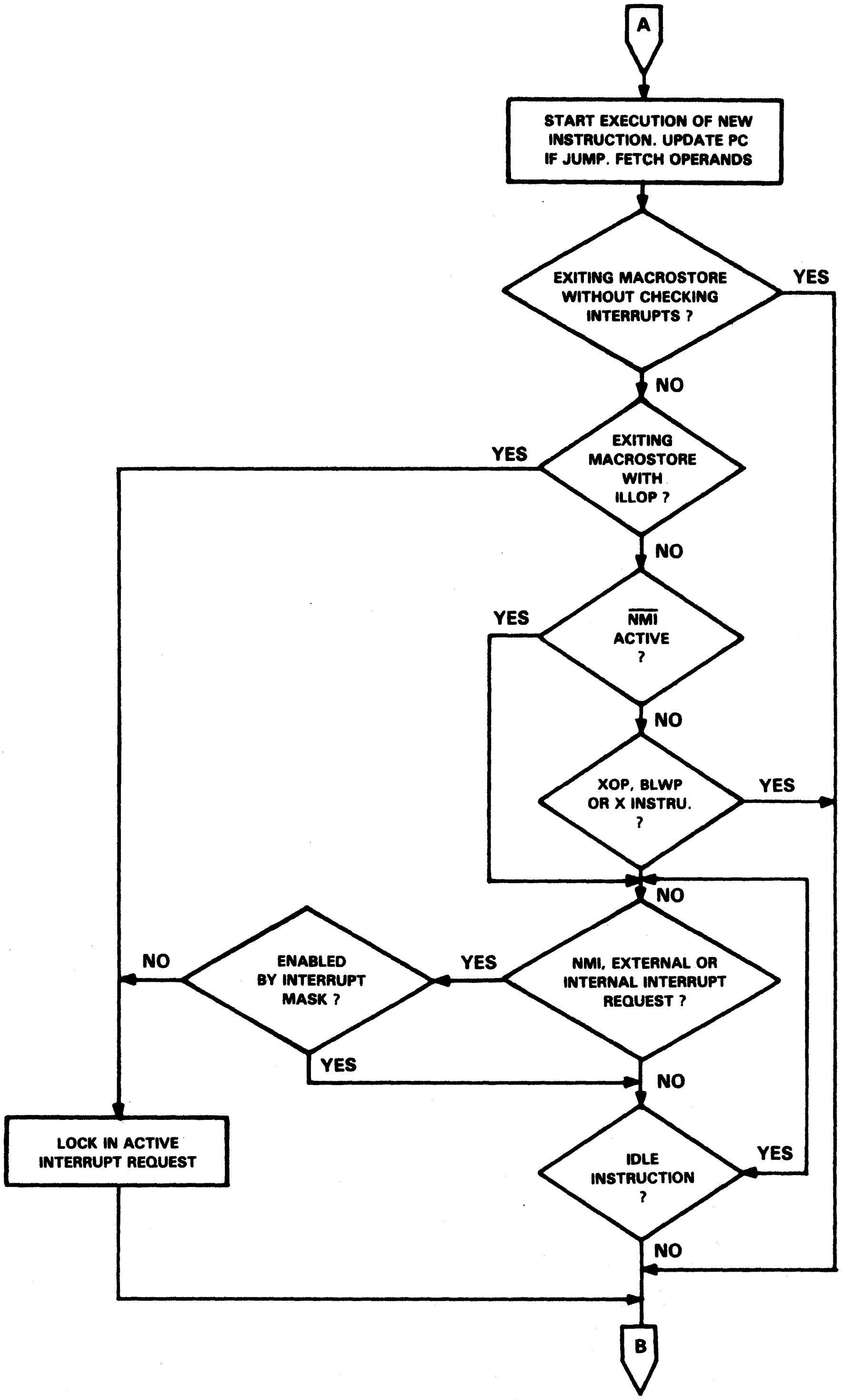

A
START EXECUTION OF NEW INSTRUCTION. UPDATE PC IF JUMP. FETCH OPERANDS
EXITING MACROSTORE WITHOUT CHECKING INTERRUPTS ?
YES
NO
EXITING MACROSTORE WITH ILLOP ?
YES
NO
$\overline{\text{NMI}}$ ACTIVE ?
YES
NO
XOP, BLWP OR X INSTRU. ?
YES
NO
NMI, EXTERNAL OR INTERNAL INTERRUPT REQUEST ?
YES
ENABLED BY INTERRUPT MASK ?
NO
YES
NO
LOCK IN ACTIVE INTERRUPT REQUEST
IDLE INSTRUCTION ?
YES
NO
B

FIGURE 4 – TMS99000 FLOWCHART (CON'D)

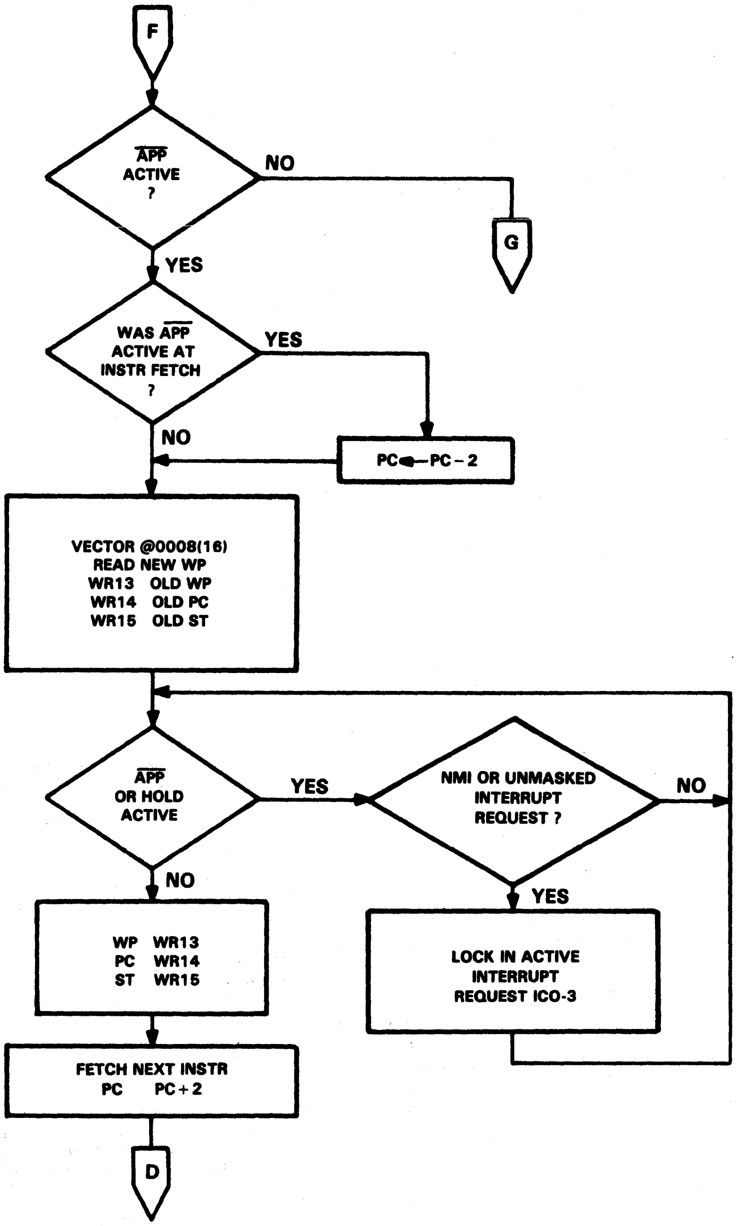

F
$\overline{\text{APP}}$ ACTIVE ?
NO
G
YES
WAS $\overline{\text{APP}}$ ACTIVE AT INSTR FETCH ?
YES
NO
PC ← PC − 2
VECTOR @0008(16)
READ NEW WP
WR13 OLD WP
WR14 OLD PC
WR15 OLD ST
$\overline{\text{APP}}$ OR HOLD ACTIVE
YES
NMI OR UNMASKED INTERRUPT REQUEST ?
NO
YES
NO
LOCK IN ACTIVE INTERRUPT REQUEST ICO-3
WP WR13
PC WR14
ST WR15
FETCH NEXT INSTR
PC PC+2
D

FIGURE 4 – TMS99000 FLOWCHART (CONT'D)

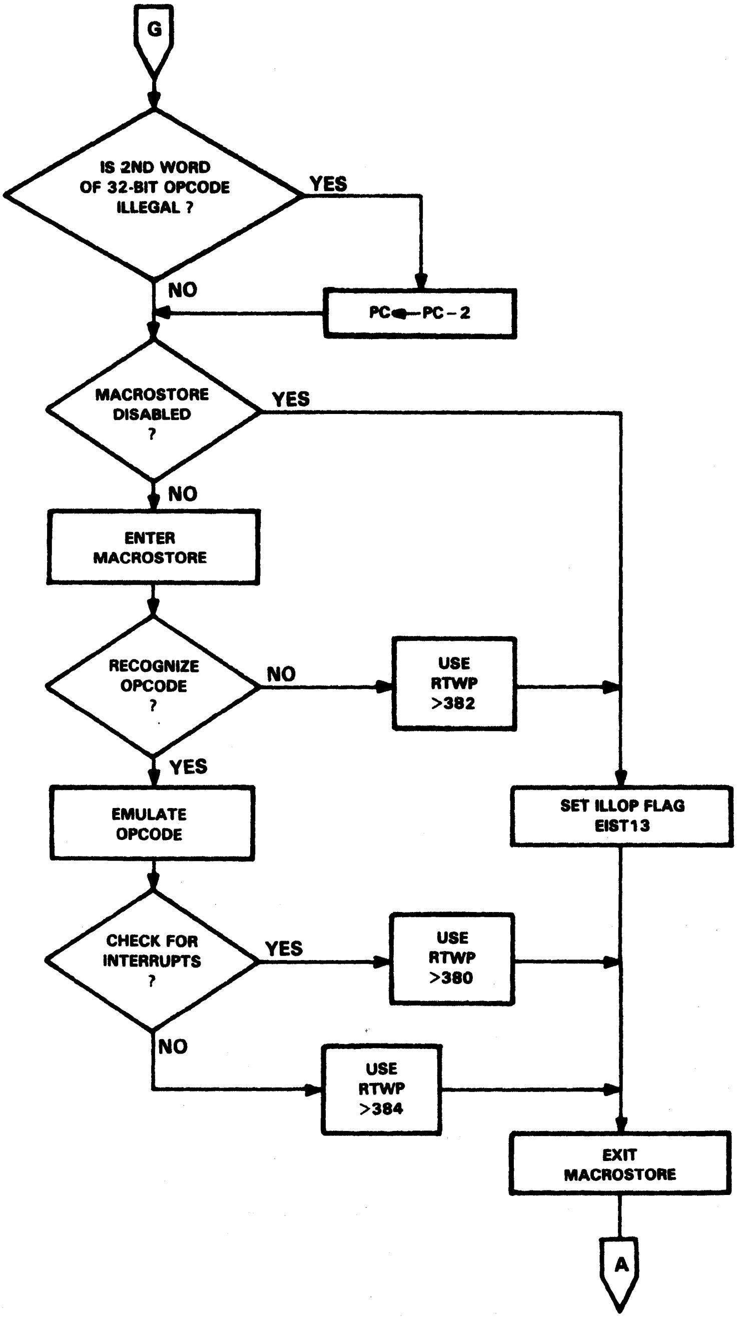


FIGURE 4 – TMS99000 FLOWCHART (CONCLUDED)

2.3 ARITHMETIC LOGIC UNIT

The arithmetic logic unit (ALU) performs all arithmetic and logical operations required during instruction execution. These operations include addition, subtraction, AND, OR, exclusive-OR and complement. A separate comparison circuit within the chip performs the logical and arithmetic comparisions needed to control bits 0, 1 and 2 (Figure 5) of the status register. Byte operations are performed in the most-significant half of the ALU. The least-significant half of the result in byte operations is left unchanged. This conveniently permits the status circuitry used for word operations to be used for byte operations as well.

2.4 INTERNAL REGISTERS

The following three internal registers are accessible to the programmer:

- Program Counter (PC)
- Status Register (ST)
- Workspace Pointer (WP)

Other internal registers that perform instruction acquisition and execution are inaccessible to the programmer.

2.4.1 Program Counter

The program counter (PC) is a 15-bit hardware register that contains the memory address (or external Macrostore address; see Section 7) of the instruction word following the currently executing instruction.

Conceptually, the PC is a 16-bit register that functions as a count-by-two counter with the least-significant bit (LSB) hardwired to 0. Since instructions are constrained to word boundaries, the processor reads the instruction word from the location pointed to by the PC, and increments the contents of the PC by two in preparation for the next instruction fetch. A program branch is performed by replacing the contents of the PC with the address of an instruction word located in memory or Macrostore. All jump, branch and context-switching instructions alter the PC in this manner.

2.4.2 Status Register

The status register (ST) is a fully-implemented 16-bit internal register whose contents signify the results of arithmetic and logical comparisons, indicate program status conditions, and supply the arithmetic fault interrupt enable and the interrupt mask to the interrupt priority circuits. Each bit position in the register represents a particular processor function or condition. Figure 5 illustrates the bit position assignments. Certain instructions, when executed, use the status register to check for a prerequisite condition; others affect the values of the bits in the register; still others load the entire status register with a new value. This last case occurs when an LST, RTWP or XOP instruction is executed. Other conditions causing a new status to be loaded are (1) an interrupt, and (2) return of system control from an attached processor (Section 8) to the host system. After the new status has been loaded, an ST bus status code is output along with bits 7 to 11 of the updated status register.

The effect of each individual instruction on the contents of the status register is described in Section 10.5. The individual status register bits are identified in Section 10.4 along with the conditions affecting each bit.

2.4.3 Workspace

A workspace is a block of 16 contiguous words in memory that contains frequently-accessed data and addresses. The location of the workspace is defined by the workspace pointer register, internal to the processor, which contains the address of the first word in the workspace.

Each word in the workspace is treated as an individual 16-bit general-purpose register. Workspace registers (WRs) contain data and addresses, and function as operand registers, accumulators, address registers and index registers. During interrupts and certain instructions, however, particular WRs are assigned the special roles described in Table 1. As indicated in the workspace map in Figure 6, all 16 WRs manipulate data and addresses, .but only WRs 1 through 15 can be used as index registers.

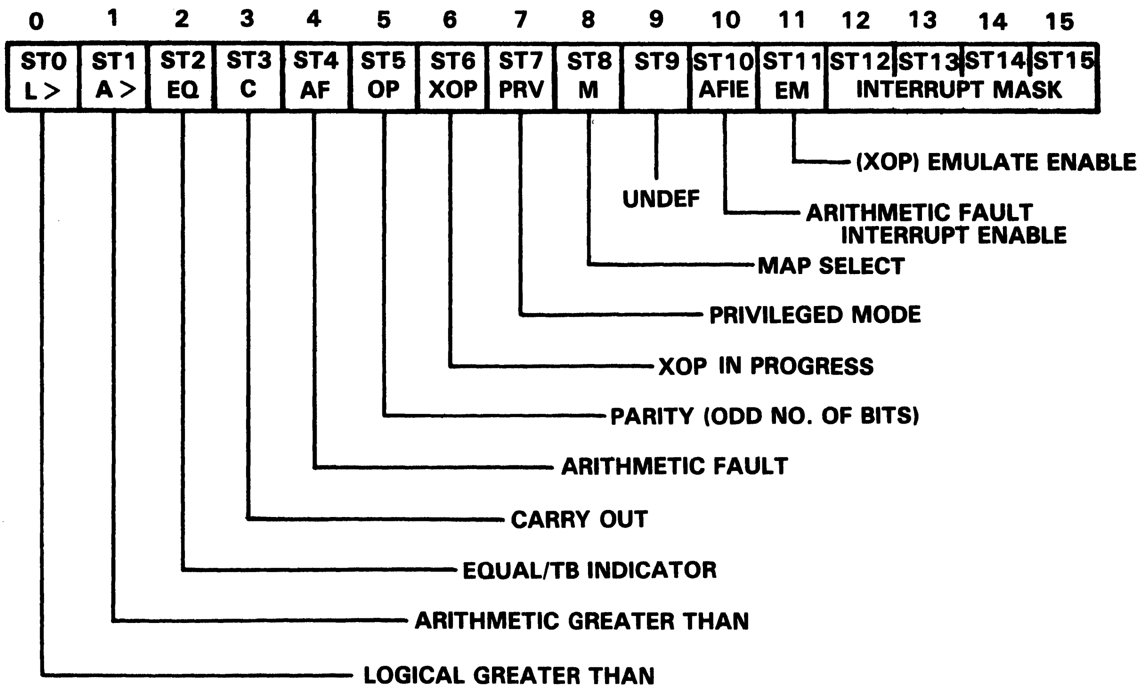


FIGURE 5 – STATUS REGISTER FIELD ASSIGNMENTS

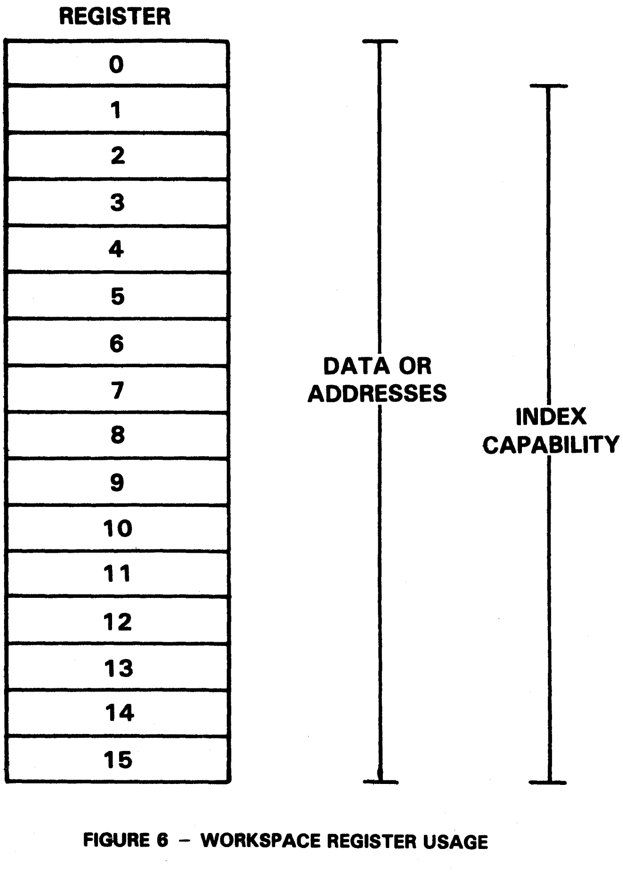
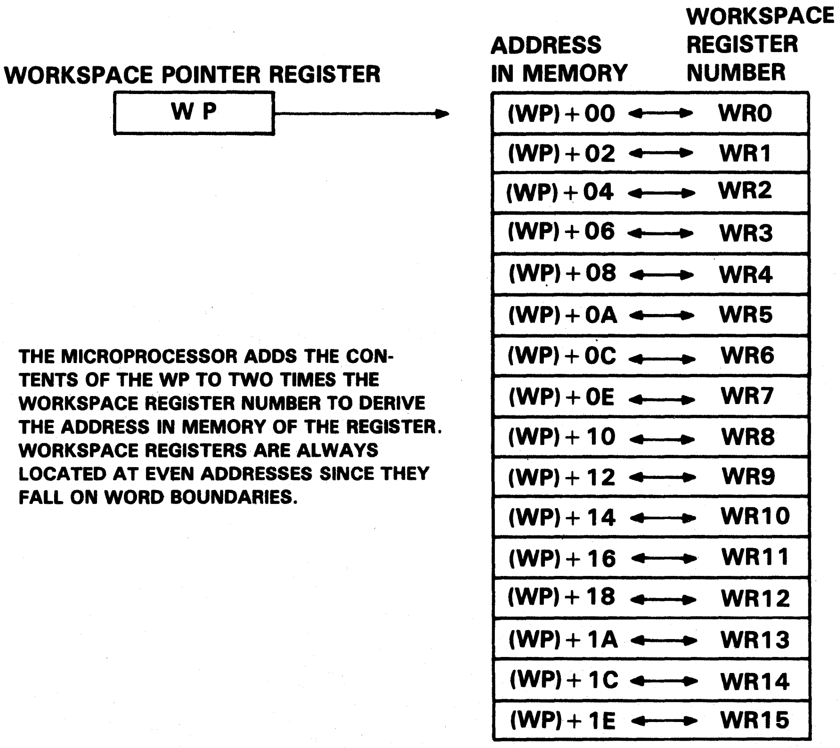


FIGURE 6 – WORKSPACE REGISTER USAGE

TABLE 1 – DEDICATED WORKSPACE REGISTERS

REGISTER NUMBER	CONTENTS	USED DURING
0	Shift count (optional)	Shift instructions (SLA, SRA, SRC, and SRL)
	Multiplicand and MSW of result	Signed multiply
	MSW of dividend and quotient	Signed divide
	MSW of floating point accumulator	Floating point operations
1	LSW of result	Signed multiply
	LSW of dividend and remainder	Signed divide
11	Return address	Branch and link instruction (BL)
	Effective address	Extended operation (XOP)
12	CRU base address	CRU instructions (SBO, SBZ, TB, LDCR, and STCR)
13	Saved WP register	Context switching (BLWP, RTWP, XOP, recognized interrupt, NMI, and RESET), external process
14	Saved PC register	Context switching (BLWP, RTWP, XOP recognized interrupt, NMI, and RESET), external process
15	Saved ST register	Context switching (BLWP, RTWP, XOP, recognized interrupt, NMI, and RESET), external process



ADDRESS IN MEMORY		WORKSPACE REGISTER NUMBER
(WP)+00	↔	WR0
(WP)+02	↔	WR1
(WP)+04	↔	WR2
(WP)+06	↔	WR3
(WP)+08	↔	WR4
(WP)+0A	↔	WR5
(WP)+0C	↔	WR6
(WP)+0E	↔	WR7
(WP)+10	↔	WR8
(WP)+12	↔	WR9
(WP)+14	↔	WR10
(WP)+16	↔	WR11
(WP)+18	↔	WR12
(WP)+1A	↔	WR13
(WP)+1C	↔	WR14
(WP)+1E	↔	WR15

THE MICROPROCESSOR ADDS THE CONTENTS OF THE WP TO TWO TIMES THE WORKSPACE REGISTER NUMBER TO DERIVE THE ADDRESS IN MEMORY OF THE REGISTER. WORKSPACE REGISTERS ARE ALWAYS LOCATED AT EVEN ADDRESSES SINCE THEY FALL ON WORD BOUNDARIES.

FIGURE 7 – WORKSPACE POINTER AND REGISTERS

The location of the workspace in memory is defined by a hardware register, internal to the processor, called the workspace pointer (WP). The WP contains the address of the first workspace register (WR0). Conceptually, the WP is a 16-bit register with the LSB hardwired to 0. As indicated in Figure 7, the memory address of WRn, n=0,1,...,15, is calculated as (WP)+2n.*

2.4.4 Context Switching

The processor's memory-resident workspace is a particularly valuable feature in applications that require frequent context switches. A context switch is a change from one program execution environment to another such as takes place during a subroutine call or an interrupt. Since the workspace registers already reside in memory, the processor performs a context switch simply by saving its three internal registers, the WP, PC and ST, in memory and fetching the new WP and PC from memory.

The processor realizes a similar time savings in returning from an interrupt or subroutine. The original context is restored by simply replacing the contents of the WP, PC and ST with the values saved in memory.

The instructions that result in a context switch include BLWP (branch and load workspace pointer), RTWP (return from subroutine or interrupt) and XOP (extended operation). A device interrupt, arithmetic fault interrupt, privileged opcode violation, illegal instruction error, a RESET or an NMI (non-maskable interrupt) also causes a context switch by forcing the processor to trap to a service routine.

2.4.5 Access of PC, ST. and WP

System control can be transferred from the processor to an external device such as an attached processor or maintenance panel. During the transfer of control, the processor writes the contents of its WP, PC and ST to memory where they can be accessed and modified by the external device. Upon return of control to the processor, the WP, PC and ST are updated with the modified values from memory. The details are presented in Section 8.

2.5 MACROSTORE

The TMS99000 addresses a 64K byte memory address space, which is logically distinct from the main memory address space. This memory space, which is called Macrostore, is logically differentiated from the main memory space through a bus status code output by the processor.

The TMS99000 has reserved the first 4K byte addresses for on-chip Macrostore memory. Of this 4K byte space, there are 1K bytes of ROM and 32 bytes of RAM implemented on the initial versions of the TMS99000. The TMS99105 does not utilize its on-chip ROM; however it does provide the 32 bytes of RAM eliminating the external RAM requirement in many cases where external Macrostore memory is provided. Other versions of the TMS99000 family provide preprogrammed functions in the on-chip Macrostore ROM (e.g. TMS99110 floating point). All members of the TMS99000 family can address external Macrostore memory for prototyping and applications requiring more than the 1K bytes of on-chip ROM.

Macrostore memory space implements added functions or instructions through emulation routines written in standard machine code. The Macrostore address space is entered through the attempted execution of a subset of the unused opcodes called macroinstructions. When attempted execution of the macroinstruction takes place, the processor traps to a specified location within the Macrostore. It is the Macrostore-resident software's responsibility to decode and perform the emulation of the function or instruction. A Macrostore memory map is shown in Figure 8. Section 7 describes the interface and use of the Macrostore memory space.

*Enclosing WP in parenthesis means that the contents of WP are being referred to.

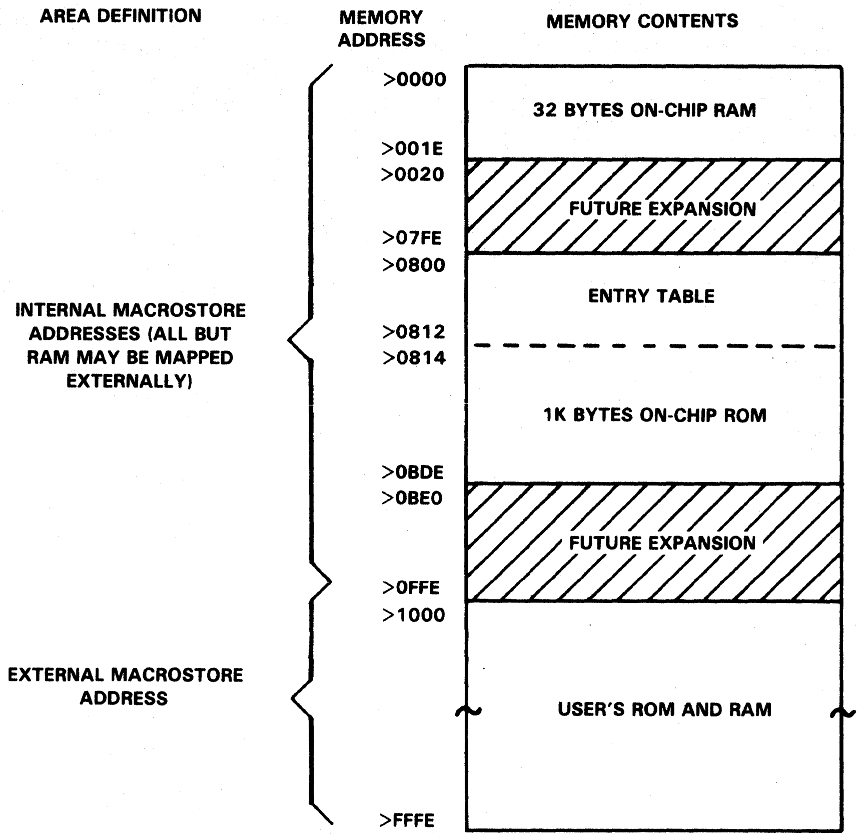


FIGURE 8 – MACROSTORE MEMORY ADDRESS SPACE

3. TMS99000 MEMORY INTERFACE

3.1 DEFINITION

The processor pin functions are described in Section 9. Several of the pins have dual or multiple functions determined by the state of the ALATCH, $\overline{\text{MEM}}$, and bus status code (BST1-BST3) outputs. Processor operations involving the transfer of data utilize the time-multiplexed address and data lines. These lines, along with the corresponding control signals, comprise the local bus interface of the processor. The local bus interface is used to perform memory, DMA, input/output, external Macrostore, and attached processor operations.

The term, bus cycle, describes the sequence of handshake operations necessary to complete the transfer of one datum over the local bus. The beginning of each bus cycle is marked by a positive ALATCH pulse, during which an address is output on the bus, and $\overline{\text{MEM}}$ and the bus status code become valid. Each particular type of bus cycle is indicated by its own unique bus status code (Table 2). A read or write operation is indicated early in the cycle by the R/$\overline{\text{W}}$ output. The R/$\overline{\text{W}}$ output acts as an early predictor of whether the AD buffers will tristate when in the data bus mode (after the falling edge of ALATCH). The R/$\overline{\text{W}}$ output is different from traditional R/$\overline{\text{W}}$ indicators in that the R/$\overline{\text{W}}$ output provides direction indication for both memory and non-memory cycles. During memory write operations, R/$\overline{\text{W}}$ remains at a low level throughout the memory cycle, During memory read operations, R/$\overline{\text{W}}$ remains at a high level throughout the memory cycle. The R/$\overline{\text{W}}$ output may be used to enable the direction on various databus buffers. The R/$\overline{\text{W}}$ output also provides an early indication of the $\overline{\text{RD}}$ output such that when R/$\overline{\text{W}}$is high at the beginning of the cycle, $\overline{\text{RD}}$ will be taken low by the processor after ALATCH goes low. If R/$\overline{\text{W}}$ is low at the beginning of the cycle, the $\overline{\text{RD}}$ output will remain high after ALATCH goes low. Following the falling edge of the ALATCH pulse, the bus is used either to perform a write operation or is forced to the high-impedance state for a read operation. The bus status, R/$\overline{\text{W}}$, and $\overline{\text{MEM}}$ outputs remain stable throughout the duration of the bus cycle, and either the $\overline{\text{WE/IOCLK}}$ or $\overline{\text{RD}}$ output may be pulsed low to perform a write or read operation.

TABLE 2 – BUS STATUS CODES

	BST				
MEM–	**1**	**2**	**3**	**NAME**	**DESCRIPTION OF BUS ACTIVITY**
L	L	L	L	SOPL	Source operand transfer with MPILCK asserted.
L	L	L	H	SOP	Source operand transfer. MPILCK is inactive.
L	L	H	L	IOP	Immediate data or second word of two-word instruction, or symbolic address.
L	L	H	H	IAQ*	Instruction acquisition. First word of instruction is fetched from memory.
L	H	L	L	DOP	Destination operand transfer.
	H	L	H	INTA	Interrupt acknowledge. Active during the WP and PC fetch for an interrupt or XOP.
L	H	H	L	WS	Workspace transfer (or multi-word transfer beginning with WR15, and Ts=0).
L	H	H	H	GM	General memory transfer.
H	L	L	L	AUMSL	Internal arithmetic-logical unit operation or macrostore access with MPILCK asserted.
H	L	L	H	AUMS	Internal arithmetic-logical unit operation or macrostore access. MPILCK is inactive.
H	L	H	L	RESET	Reset. The $\overline{\mathrm{RESET}}$ input is pulled low.
H	L	H	H	IO	I/O transfer
H	H	L	L	WP	Workspace pointer update due to BLWP, RTWP, LWP, XOP, APP entry, APP exit or interrupt. The new workspace pointer is on the address bus.
H	H	L	H	ST	Status register update due to LST, RTWP, XOP, APP exit or interrupt. Bits 7-11 of the new status are on the address bus. This occurs prior to the fetch of the next instruction.
H	H	H	L	MID	Macroinstruction detected. $\overline{\mathrm{APP}}$ is sampled when READY is high.
H	H	H	H	HOLDA	Hold acknowledge.

*Due to opcode prefetch, IAQ for the next instruction may be output before the result of the current instruction is stored.

During bus cycles dedicated to internal functions, the $\overline{\mathrm{RD}}$ and $\overline{\mathrm{WE}}/\overline{\mathrm{IOCLK}}$ outputs remain high and R/$\overline{\mathrm{W}}$ goes low: no transfer of data takes place, although a bus status code is output. During these operations, the activity of the ALATCH and address-data lines is as described in the previous paragraph.

For convenience, reference will occasionally be made to the "address bus" and "data bus" as if they were separate lines. The reader should remember that address and data are, in fact, multiplexed over the same physical lines.

The basic time unit of the local bus interface is the machine state, which has a duration of one CLKOUT period. A bus cycle minimally requires one machine state to complete but may be extended by some integral number of additional machine states.

Bus cycles can be extended by the READY input signal. READY is manipulated by external logic to permit the processor to work with slow memory or I/O devices. The additional machine states generated by the READY signal are called wait states. It should be noted that wait states may be generated even during internal ALU cycles as indicated by the bus status codes.

Three types of bus cycle are distinguished: memory, I/O and internal. During a memory or I/O cycle, a data transfer takes place on the local bus accompanied by either the $\overline{\mathrm{WE}}/\overline{\mathrm{IOCLK}}$ output signal. During all cycles R/$\overline{\mathrm{W}}$ also is output to give an early indication of read/write at the start of the cycle and continues to be active until the end of the bus cycle. Either $\overline{\mathrm{RD}}$ or $\overline{\mathrm{WE}}/\overline{\mathrm{IOCLK}}$ is active during an internal cycle involving an access of external macrostore. During internal machine cycles which are not Macrostore cycles (as defined by the bus status codes AUMSL and AUMS), the $\overline{\mathrm{RD}}$ and $\overline{\mathrm{WE}}/\overline{\mathrm{IOCLK}}$ outputs remain inactive high. Memory and non-I/O cycles have a minimum duration of one machine state, and a I/O cycle has a minimum duration of two machine states. A memory, I/O or internal cycle can be extended by an arbitrary number of wait states by pulling the READY input low. Note that because internal cycles can be wait-stated, care must be exercised in the design of external READY control logic to avoid wait-stating internal cycles that are not Macrostore cycles.

3.2 MEMORY INTERFACE

The signals used in the interface to system memory are shown in Figure 9.

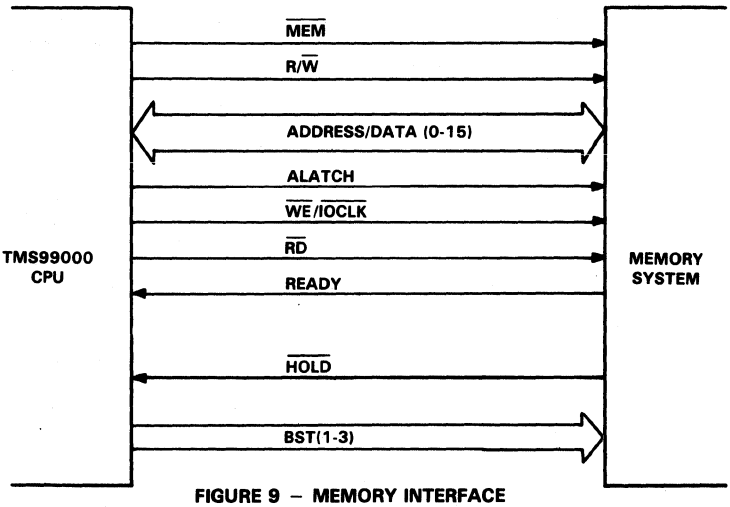
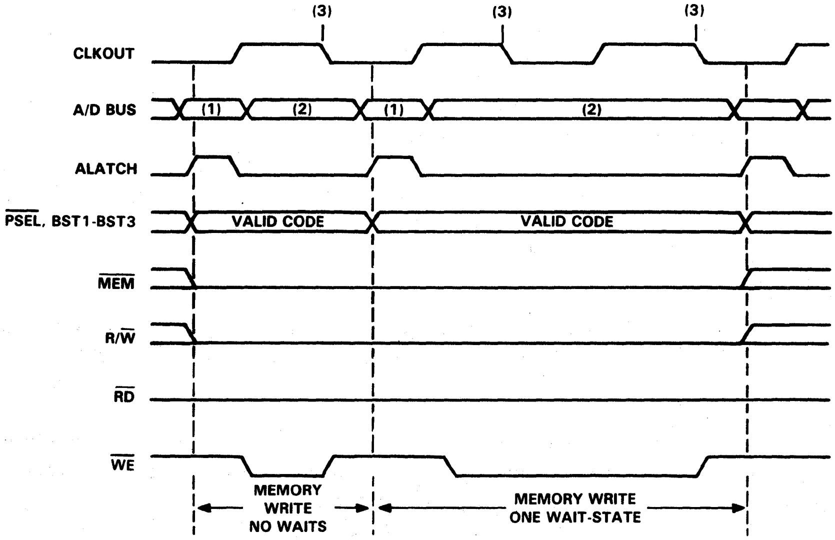


FIGURE 9 – MEMORY INTERFACE

3.2.1 Memory Write Operations

The timing for a memory write cycle is shown in Figure 10. At the beginning of the cycle, the processor asserts ALATCH, outputs the address and PSEL on the address-data lines, and pulls MEM low. Concurrent with MEM going low, R/W goes low to give an early indication of a memory write cycle. The CPU then pulls ALATCH low, outputs the data word on the address-data lines, and asserts WE/IOCLK. The cycle may be extended by wait states using the READY signal, as described in Section 3.2.5.

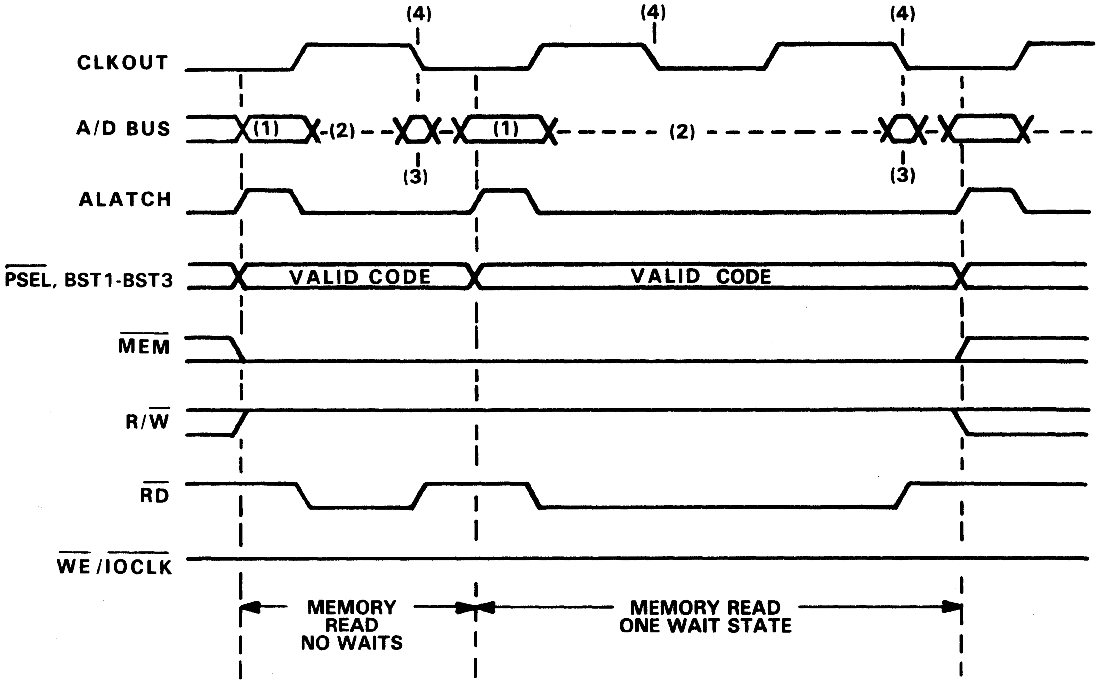


NOTES:
(1) Address and PSEL are valid.
(2) Memory write data valid.
(3) READY is sampled at this time.

FIGURE 10 – MEMORY WRITE CYCLE OPERATION

3.2.2 Memory Read Operations

The timing for a memory read cycle is shown in Figure 11. At the beginning of the cycle, the processor asserts ALATCH, outputs the address and $\overline{\text{PSEL}}$ on the address-data lines, and pulls $\overline{\text{MEM}}$ low. The R/$\overline{\text{W}}$ output goes to a high level to indicate that the cycle is to be a memory read operation. The processor then pulls ALATCH low, forces the address-data lines into the high-impedance state, and pulls $\overline{\text{RD}}$ low to enable the read data from memory onto the address-data lines. The cycle may be extended by wait states using the READY signal, as described in Section 3.2.5.



NOTES:
(1) Address and $\overline{\text{PSEL}}$ are valid.
(2) Bus is in input mode (drivers are tristated).
(3) Memory read data must be valid at indicated CLKOUT edge.
(4) READY is sampled at this time.

FIGURE 11 – MEMORY READ CYCLE OPERATION

3.2.3 Extended Memory Addressing

Several techniques are available for extending the address reach of the processor. These techniques use the $\overline{\text{PSEL}}$ and bus status codes (BST1-BST3) to provide for extended address reach by defining additional 64K byte pages of memory based on information output by the processor during every memory cycle.

3.2.3.1 Memory Paging

Status bit 8 of the status register is inverted and multiplexed on the $\overline{\text{PSEL}}$/DO/OUT pin. $\overline{\text{PSEL}}$ may be used as a 17th address bit to select between two pages of 64K bytes for a total address reach of 128K bytes of physical memory.

The $\overline{\text{PSEL}}$ signal output occurs concurrently with the memory address when ALATCH is active high. The following instructions force the $\overline{\text{PSEL}}$ output to the high state regardless of the value of ST8 of the status register:

- RTWP return from interrupt
- XOP extended operations (software trap)
- All interrupts
- All I/O instructions

In addition, the LST (load status) instruction can modify the $\overline{\text{PSEL}}$ output if the state of ST8 of the status register is changed by the instruction. The long distance source/destination instructions (LDS, LDD) cause the $\overline{\text{PSEL}}$ to be inverted from the previous state during the source or destination access by the instruction following LDS or LDD, respectively. (see Section B.3).

3.2.3.2 *Functional Segmentation*

In addition to paging capability using the $\overline{\text{PSEL}}$ output, memory may be segmented functionally into an instruction segment and a data segment. Referring to Table 2, the bus status codes IOP and IAQ may be decoded to create a segment-select line for differentiating between references to the instruction segment and the data segment. Note that BST3 is a "don't care" during the decode operation. The decoding necessary to distinguish between references to data and instruction segments is shown in Figure 12. Figure 13 illustrates the hierarchy of a 256K byte physical memory system utilizing the memory paging and functional segmentation techniques.

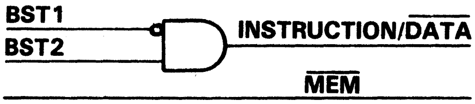


FIGURE 12 – FUNCTIONAL SEGMENTATION LOGIC

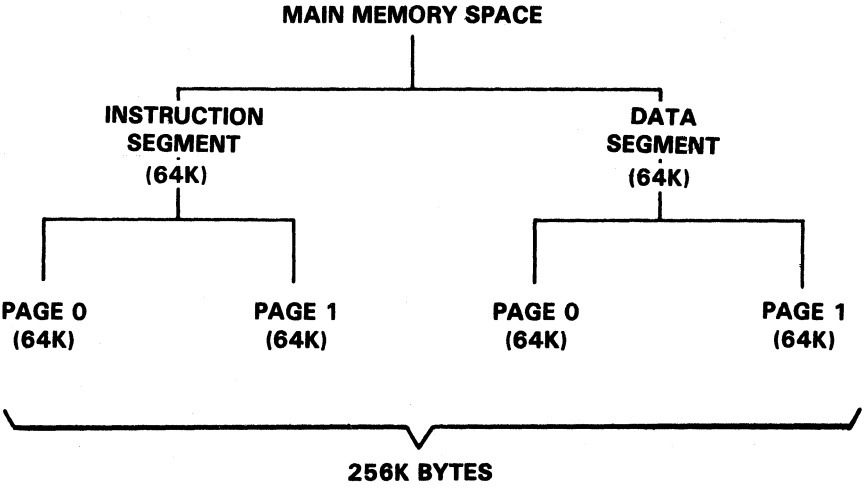


FIGURE 13 – TMS99000 EXTENDED ADDRESSING

3.2.3.3 *Memory Mapping Techniques*

The TMS99000 may utilize the TIM99610 memory mapper (SN74LS610) device to extend the address reach of the processor to 16 megabytes. The TIM99610 device contains 16 12-bit map registers, which are selected by the TMS99000's four most-significant address lines. These 12 bits are output from the TIM99610 and appended to the address bus as the most-significant address lines. Thus, mapped pages may reside on any 4K-byte address boundary.

The PSEL output may be used to enable/disable the operation of the memory mapper. If PSEL is connected to the MM pin of the mapper circuit, the mapping of the internal map registers occurs only when PSEL is low. When PSEL is inactive high, the four address bits present on the register-select inputs are passed through to the outputs unchanged. This allows for correct operation when interrupt or XOP (extended operations) vectors are fetched from predefined locations.

The TMS99110 contains two instructions which are designed to facilitate operation with a TIM99610 memory mapper. They are Long Distance Source (LDS) and Long Distance Destination (LDD). These instructions are described in more detail in the TMS99110 supplement (Appendix B). The LDS and LDD instructions invert the PSEL output when performing source and destination operand fetches of the following instruction. This allows an instruction to reach operands outside the boundaries of the current page. Figure 14 illustrates the interface between a TMS99000 and the TIM99610 memory mapper.

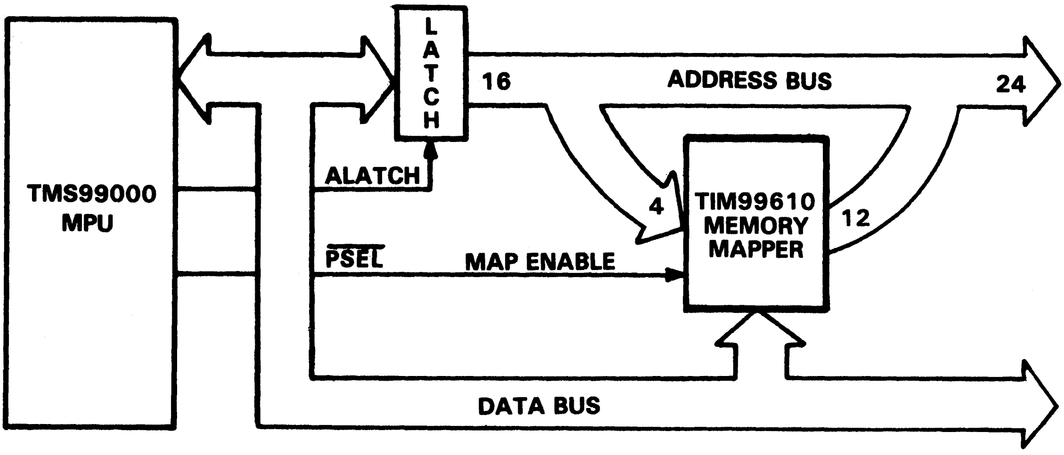
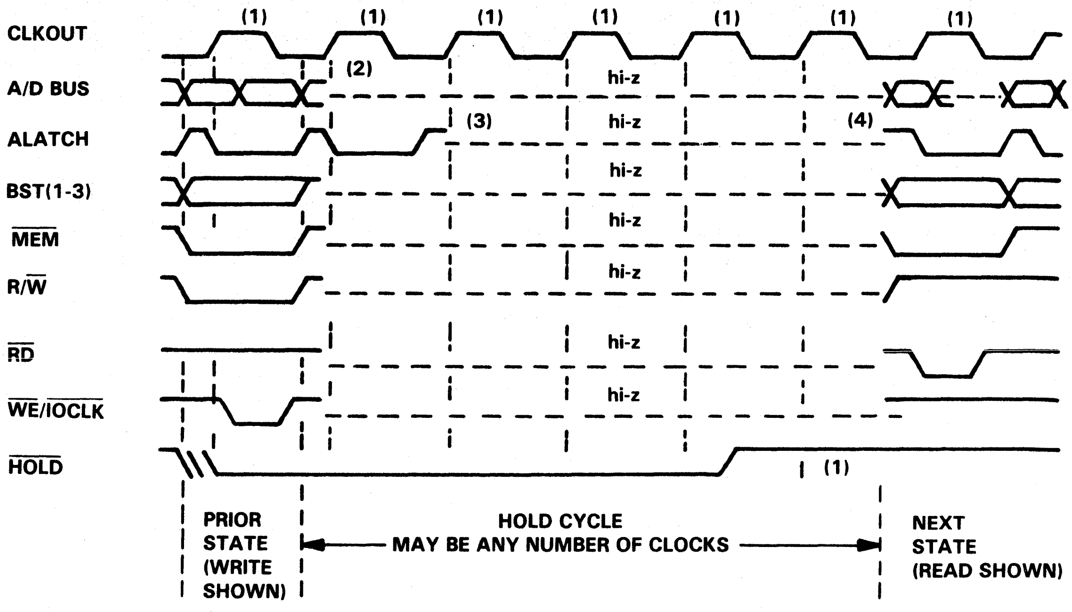


FIGURE 14 – TMS99105A OR TMS99110A TO TIM99610 MEMORY MAPPER INTERFACE

3.2.4 Direct Memory Access

The processor provides the signals necessary to allow DMA devices to directly transfer information to and from the system memory. To gain control of the local bus interface, the DMA device sends a hold request to the processor by pulling the processor HOLD input low.

The timing for the hold cycle is presented in Figure 15. Assume that HOLD is pulled low during a memory write cycle, as indicated in the example of Figure 15. As soon as the ongoing cycle is complete, the processor responds to the HOLD signal by outputting a HOLDA bus status code (MEM, R/W and BST1-BST3 are all driven high); this signals its impending surrender of the local bus to the DMA device. The bus status code is held only for a quarter state, long enough to be latched externally on the falling edge of ALATCH. As soon as ALATCH has made its high-to-low transition, the following output signals are forced to the high-impedance state: MEM, R/W, BST1-BST3, WE/IOCLK, RD and the address-data lines. At the beginning of the next machine state, the ALATCH signal is driven high for a quarter state, after which it also is forced to high impedance. These lines remain in the high impedance state for the duration of the hold cycle. The CLKOUT output line, on the other hand, remains active through the hold cycle. The DMA device takes control of the local bus and performs its transfer or transfers of data to or from main memory. When the DMA device has completed its transfers, it deactivates the HOLD signal. The processor responds by removing the HOLDA bus status code, and leaves the hold state to resume processing.



NOTES:

(1) CLKOUT edge at which HOLD is sampled.

(2) Tristate all outputs except ALATCH as follows:

- BST(1-3) and R/W are first driven high to indicate hold acknowledge, and then tristated.
- MEM, RD, WE, and R/W are first driven high, and then tristated.
- The data bus is tristated as is.

(3) ALATCH is first driven high, and then tristated.

(4) All outputs become active again.

FIGURE 15 – MEMORY CYCLE – DMA HOLD OPERATION

The processor samples HOLD at the falling edge of each CLKOUT pulse. Sampling of HOLD occurs even while the MPILCK (multiprocessor interlock) bus status code is being output (Section 3.4.4) in order to reduce worst-case DMA latency.

If HOLD is asserted at the beginning of a reset operation, the processor requests no memory cycles until HOLD is removed. This permits automatic DMA loading of memory after power up. When HOLD and RESET are pulled low at the same clock edge, the RESET bus status code will be output prior to the HOLDA bus status code and for as long as RESET remains active low.

If the READY input signal is low when HOLD is released, the hold cycle is extended with wait states until READY is allowed to go high.

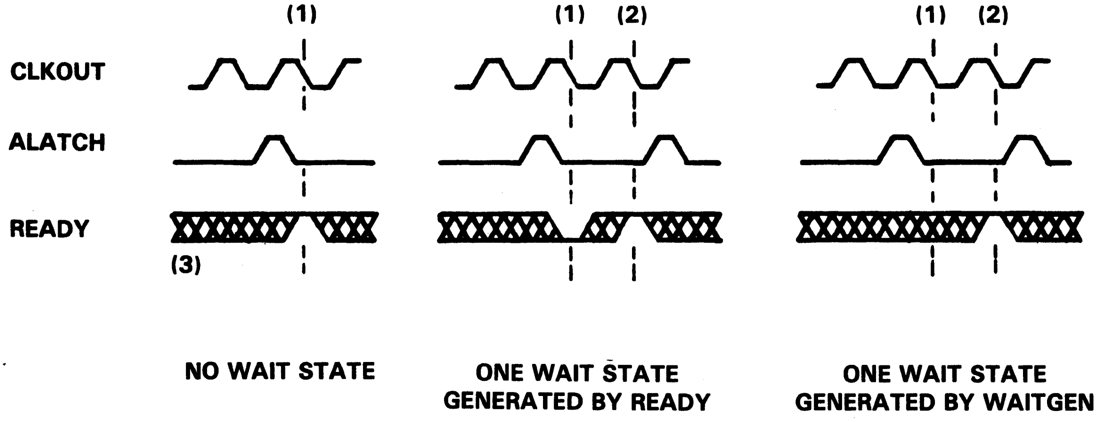
3.2.5 Memory Wait-State Generation

The READY input is held low to extend memory, I/O, and internal bus cycles by an arbitrary number of wait states. Wait states continue to be generated until READY is released (i.e., allowed to go high). Wait state generation for I/O cycles is presented in Section 5.

READY is low during the first machine state of a memory cycle, however, the cycle is extended by one wait-state. If READY continues to be held low, the memory cycle is extended by additional wait-states until READY goes high.

External Macrostore accesses are treated as a special type of internal cycle. These cycles can be extended with wait-states by pulling READY low (see Section 7).

The timing for wait state generation during memory cycles is shown in Figure 16. This same technique may be used for machine cycles which are neither memory nor I/O cycles (i.e. internal cycles). Thus care must be taken when designing circuitry controlling the READY input. As indicated in the figure, READY is sampled at the falling edge of CLKOUT.


(1)
(1) (2)
(1) (2)
CLKOUT
ALATCH
READY
(3)
NO WAIT STATE
ONE WAIT STATE GENERATED BY READY
ONE WAIT STATE GENERATED BY WAITGEN

NOTES:
(1) First sample time of READY during bus cycle.
(2) Second sample time of READY during bus cycle. Additional wait states are generated by keeping READY low at this and subsequent sample times.
(3) XXXXXX denotes don't care.
(4) READY is sampled on non-memory as well as memory cycles.

FIGURE 16 – WAIT-STATE GENERATION FOR MEMORY BUS CYCLES

3.3 PROCESSOR INTERNAL CYCLE INDICATION

The bus status code output by the processor distinguishes internal cycles from memory and I/O cycles. Referring to Table 2, the AUMSL, AUMS, RESET, WP, ST, MID, and HOLDA codes indicate the particular type of internal cycle in progress. The AUMS or AUMSL code is output during accesses of external Macrostore.

The MPILCK (multiprocessor interlock) condition is signified by BST = 000 and can remain in effect during an internal cycle, as indicated by the AUMSL bus status code.

Each internal cycle begins with an ALATCH pulse. During an internal cycle, a low READY signal will generate wait-states. Activity on the local bus interface during internal cycles is discussed in Section 10.6.4.

3.4 Applicable Bus Status Codes

The bus status codes that are relevant to the interface between the processor and system memory are described in the following paragraphs. Each bus cycle – memory, I/O or internal – is accompanied by a bus status code consisting of the $\overline{\text{MEM}}$ and BST1-BST3 output signals, as indicated in Table 2. The bus status code for each bus cycle becomes valid during the ALATCH pulse at the beginning of the cycle and remains valid through the remainder of the cycle. An access of internal Macrostore is classified as a special type of internal cycle and is accompanied by the AUMS or AUMSL bus status codes.

3.4.1 Memory Read Cycle Codes

For all memory read cycles, $\overline{\text{MEM}}$ is active low. The bus status codes (from Table 2) differentiate the following types of memory read cycles:

- IAQ – instruction acquisition
- IOP – fetch data from the instruction stream (immediate operands, symbolic addresses, or second word of a two-word instruction)
- SOP – source operand
- SOPL – source operand with MPILCK asserted
- DOP – destination operand
- INTA – fetch of interrupt or XOP trap vector (WP and PC), including NMI and reset
- WS – workspace (Note that the WS bus status code will occur only when workspace register addressing is used. When the workspace is accessed via other addressing modes (i.e., symbolic), the WS code will not be output.)
- GM – general memory

During an indivisible (semaphore) operation, the MPILCK bus status code becomes active at the start of the source operand read cycle (Section 3.4.4).

3.4.2 Memory Write Cycle Codes

For all memory write cycles, $\overline{\text{MEM}}$ is active low. The bus status codes differentiate the following types of memory write cycles:

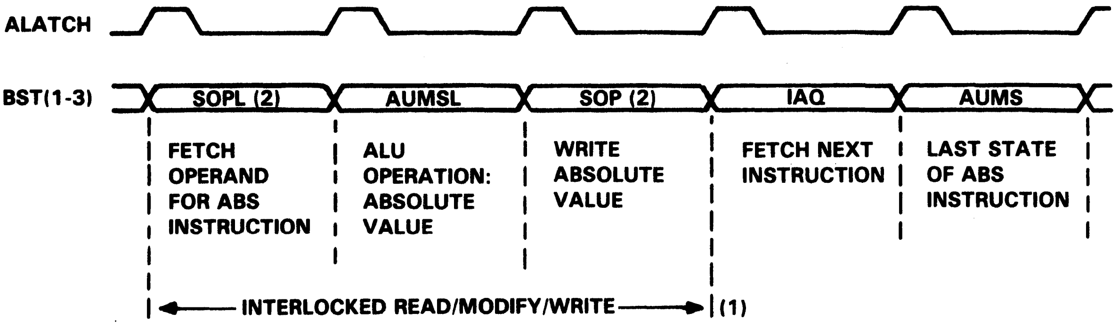
- SOP – source operand
- DOP – destination operand
- WS — workspace (Note that the WS bus status code will occur only when workspace register addressing is used. When the workspace is accessed via other addressing modes (i.e. symbolic), the WS code will not be output.)
- GM – general memory

3.4.3 Hold Acknowledge Code – HOLDA

The processor outputs the HOLDA bus status code (Table 2) upon relinquishing the local bus in response to an active $\overline{\text{HOLD}}$ or $\overline{\text{APP}}$ input signal.

3.4.4 Multiprocessor Interlock Code – MPILCK

The MPILCK bus status code provides a means for implementing an indivisible test-and-set mechanism. Such a mechanism is required to insure system integrity in applications in which multiple processors communicate by means of semaphores located in shared memory. Whenever the processor outputs the MPILCK code, external logic inhibits memory accesses by the other processors in the system. The MPILCK signal is indicated by BST1-BST3=000 (refer to Table 2), and is output during execution of the ABS (absolute value), TSMB (test and set), and TCMB (test and clear) instructions, as shown in Figure 17. The MPILCK code becomes valid during the source operand fetch (indicated by the SOPL code in Table 2) remains active through the internal cycle (AUMSL), and is removed as the next cycle (either an SOP or WS) begins.


ALATCH
BST(1-3)
SOPL (2)
AUMSL
SOP (2)
IAQ
AUMS
FETCH OPERAND FOR ABS INSTRUCTION
ALU OPERATION: ABSOLUTE VALUE
WRITE ABSOLUTE VALUE
FETCH NEXT INSTRUCTION
LAST STATE OF ABS INSTRUCTION
INTERLOCKED READ/MODIFY/WRITE
(1)

NOTES:

(1) The SOPL or AUMSL code indicates that the lock is in effect through the next bus cycle.

(2) If Ts=0 (register source operand), an interlock will not be performed. WS will appear instead of SOPL or SOP.

FIGURE 17 – MULTIPROCESSOR INTERLOCK TIMING – ABS, TSMB, TCMB INSTRUCTIONS

The MPILCK code is not output if the source operand for an ABS, TSMB or TCMB instruction is located in the workspace. In this case, the SOPL-AUMSL-SOP bus status code sequence described above is replaced by WS-AUMS-WS.

The TMS99000 does not inhibit the sampling of $\overline{\text{HOLD}}$ while MPILCK is active; the processor will respond to the $\overline{\text{HOLD}}$ signal by replacing its MPILCK bus status code with the HOLDA code and entering hold. Using the MPILCK signal to inhibit contention for shared memory is therefore not sufficient to insure the integrity of systems which allow DMA devices to modify semaphore locations. In such systems, DMA devices must monitor MPILCK to avoid asserting $\overline{\text{HOLD}}$ during indivisible operations.

3.4.5 Macrostore Accesses

The AUMS or AUMSL (arithmetic logical unit or Macrostore) bus status code is used to indicate either a Macrostore access or an internal processing cycle; i.e., the same status code is used for both types of operation. The AUMS or AUMSL bus status code serves to distinguish accesses of external Macrostore from I/O accesses or accesses of the user's main memory. A complete description of Macrostore accesses is given in Section 7.

4. INTERRUPT STRUCTURE

4.1 TMS99000 INTERRUPT STRUCTURE

The TMS99000 provides 16 interrupt levels, each supported by its own trap vector located in memory. The trap vector for each interrupt level is a two-word structure containing the WP (first word) and PC (second word) values of the service routine. When an interrupt occurs, the ensuing context change causes the processor's internal PC and WP registers to be loaded with the values from the corresponding trap vector. The locations of the trap vectors for the 16 interrupt levels are given in Table 3. Interrupt level 0 is the highest priority, and level 15 the lowest. The reset function uses level 0. Level 2 is reserved for the illegal instruction trap, the privileged opcode violation trap, and (at the user's option) the arithmetic fault trap. The occurrence of the arithmetic fault and privileged violation interrupts (when unmasked) causes the external maskable interrupts to be ignored until after the context switch for these interrupts has occurred. Levels 1 through 15 can be used for external device interrupts; level 0 can also be used for external interrupts if external hardware is provided (Section 4.2).

TABLE 3 – INTERRUPT LEVEL DATA

INTERRUPT LEVEL	VECTOR LOCATION (MEMORY ADDRESS IN HEX)	DEVICE ASSIGNMENT	MASK VALUES TO ENABLE (ST12 THRU ST15)	VALUE MASK SET TO UPON TAKING THE INTERRUPT (ST12-ST15)
RESET	0000	External	0 through F	0
ILLOP	0008	Internal	(see Note 4)	1
NMI	FFFC	External	0 through F	0
ARITHMETIC FAULT	0008	Internal	(see Note 2, 3)	1
PRIVILEGED VIOLATION	0008	Internal	(see Note 3)	1
0	0000	External	0 through F (see Note 1)	0
1	0004	External device	1 through F	0
2	0008	External device	2 through F	1
3	000C	"	3 through F	2
4	0010	"	4 through F	3
5	0014	"	5 through F	4
6	0018	"	6 through F	5
7	001C	"	7 through F	6
8	0020	"	8 through F	7
9	0024	"	9 through F	8
A	0028	"	A through F	9
B	002C	"	B through F	A
C	0030	"	C through F	B
D	0034	"	D through F	C
E	0038	"	E and F	D
(Lowest priority) F	003C	External device	F only	E

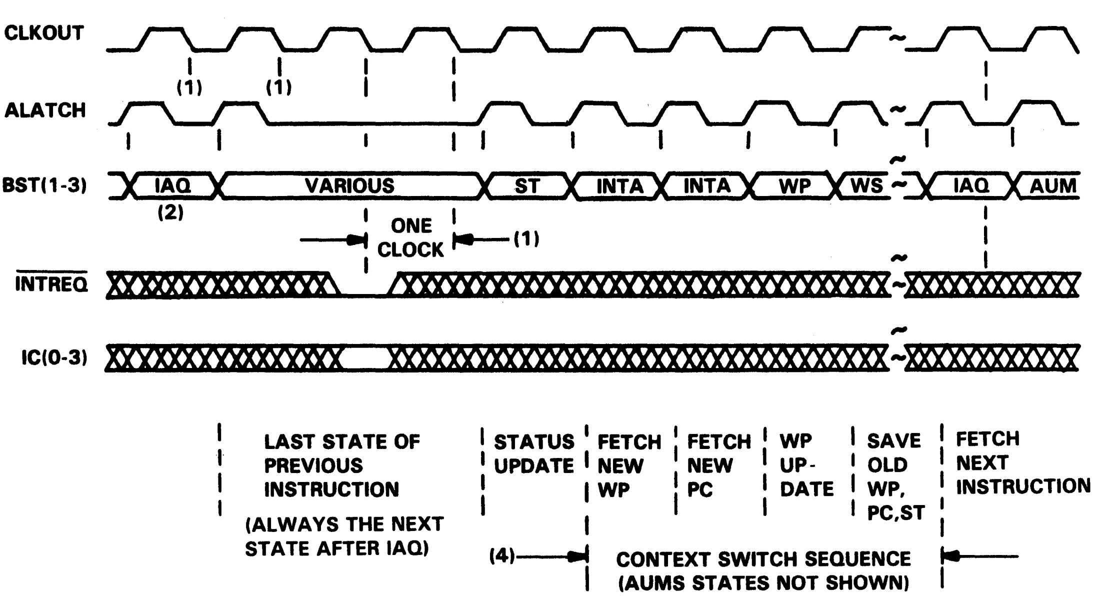
NOTES: (1) Level 0 cannot be disabled.

(2) Arithmetic fault interrupt is generated internal to the Alpha and is enabled/disabled by bit 10 of the status register.

(3) The occurrence of the arithmetic fault and privileged violation interrupts (when unmasked) causes the external maskable interrupts to be ignored until after the context switch for these interrupts has occurred.

(4) The ILLOP (illegal instruction) interrupt is generated internal to the 99000 and cannot be disabled by the interrupt mask.

External device interrupt requests are transmitted to the processor through the $\overline{\text{INTREQ}}$ and IC0-IC3 input pins. The interrupt level, in the range 0 to 15, is encoded on the four IC (interrupt code) lines, and the interrupt request is generated by pulling $\overline{\text{INTREQ}}$ low. Figure 18 shows the timing for the external interrupt interface. Activation of the $\overline{\text{INTREQ}}$ input causes the processor to compare the interrupt code, IC0-IC3, with the interrupt mask in bits 12 through 15 of the status register. If the level of the pending interrupt is less than or equal to the enabling mask level (higher or equal priority interrupt), the processor recognizes the interrupt and initiates a context switch as soon as the current instruction completes execution. The processor then fetches the new context (WP and PC) from the appropriate trap vector and at the same time forces the $\overline{\text{PSEL}}$ output high, as indicated in Figure 18. During the fetch of the new WP and PC values, the INTA (interrupt acknowledge) bus status code (Table 2) is output. Next, the previous context, consisting of the WP, PC and ST values from the interrupted program is stored in WRs 13, 14 and 15, respectively, of the new workspace. Status bits 7 through 11 are cleared to insure that the arithmetic fault interrupt enable (ST10), map enable (ST8), and privileged mode (ST7) status bits are not carried over from the interrupted program. Next, the processor forces the interrupt mask to a value that is one less than the level of the interrupt being serviced, except in the case of a level 0 interrupt, for which the mask is set to all zeros. This mechanism insures that the service routine for an external interrupt of level 1 through 15 will be interrupted only in the event that a higher-priority interrupt request is received. Upon switching to the service routine, the processor inhibits further interrupts until the first instruction of the service routine has been executed.



NOTES:

(1) INTREQ and IC(0-3) are first sampled during the IAQ cycle, but if wait states occur in the cycle that follows the IAQ cycle, samples will continue to be taken until one clock before the end of that cycle. Only the last sample taken is examined by the processor's interrupt logic to determine whether to take the interrupt. Sampling occurs at the high-to-low transition of CLKOUT.

(2) The prefetched instruction will be discarded when the interrupt is accepted.

(3) INTREQ is not sampled during the first instruction fetch following the interrupt context switch sequence.

(4) Bus cycles, during which the AUMS bus status code is output, are omitted from the figure for simplicity.

FIGURE 18 – INTERRUPT SEQUENCE

In order to insure recognition of an interrupt request, the request should remain active until acknowledged either by software in the interrupt service routine or by hardware keyed to the INTA bus status code. If a software interrupt acknowledgment is used, the interrupt service routine must reset the interrupt request before the routine is completed. If hardware interrupt acknowledgment is used, the interrupting device must monitor four bits of the address bus (A10-A13) to determine which interrupt level is being acknowledged. In the event that an interrupt level is shared by more than one device, a hardware or software priority scheme must identify the interrupting device.

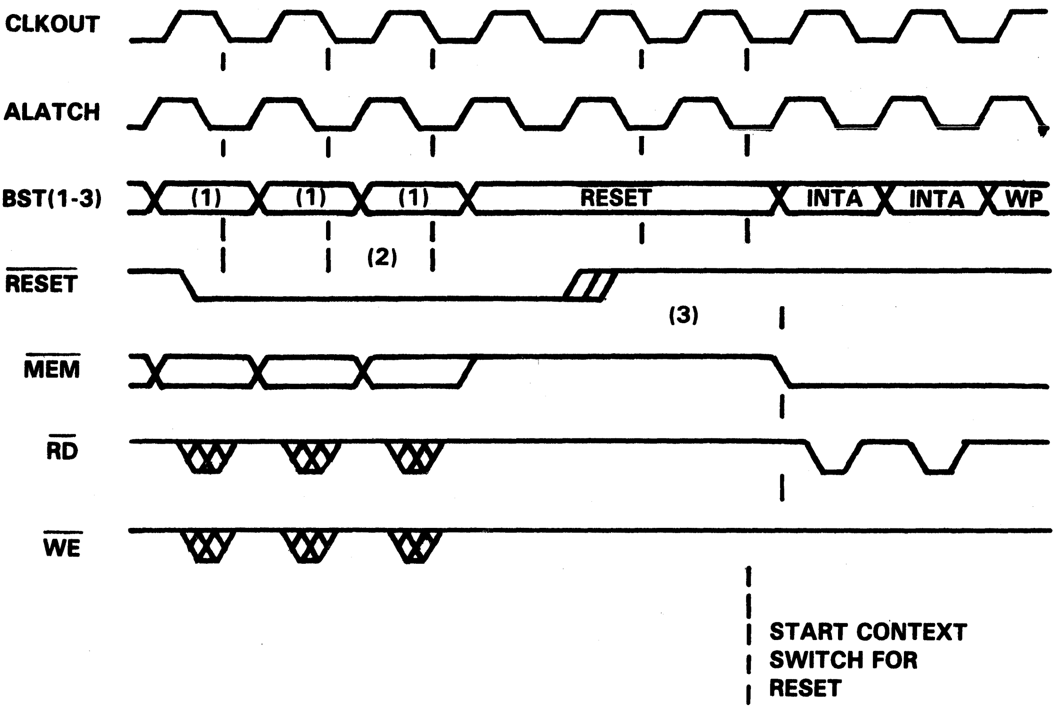
The interrupt code on the IC0-IC3 inputs will continue to be sampled as long as INTREQ remains active low. If the code specifies an interrupt level that is disabled initially by the interrupt mask value, the INTREQ input can be held low until the processor alters the mask to a value that allows the interrupt request to be recognized. The external interrupt interface is synchronous. The IC0-IC3 inputs must be stable during the falling edge of CLKOUT at which time they are sampled.

The interrupt vector is typically read from memory, but the system can be constructed so that the interrupting device itself supplies the interrupt vector via the memory bus. In this case, a hardware decoder triggered by the INTA bus status code and address bits A10 to A14 (these indicate the interrupt level and whether the WP or PC value is being read) enables the interrupting device (and disables memory) when the processor fetches the interrupt vectors.

Should the service routine for one interrupt level be interrupted by another interrupt of higher priority, a second context switch occurs to service the higher-priority interrupt. When the service routine for the higher-priority interrupt is complete, an RTWP instruction is executed to resume processing of the lower-priority interrupt. All interrupt service routines should terminate with the RTWP instruction to restore the context of the interrupted program.

4.2 INTERRUPT LEVEL 0 AND RESET

The level 0 trap vector is utilized by both reset function and the level 0 external interrupt. The reset function is activated by pulling the $\overline{\text{RESET}}$ input low. As indicated in Figure 19, the processor samples $\overline{\text{RESET}}$ on each high-to-low transition of CLKOUT. The $\overline{\text{RESET}}$ signal causes the processor to cease instruction execution at the end of the current bus cycle, and the $\overline{\text{WE/IOCLK}}$, $\overline{\text{RD}}$, and $\overline{\text{MEM}}$ signals are forced inactive high and R/$\overline{\text{W}}$ is forced low indicating the AD bus will tristate. The processor remains in this state until $\overline{\text{RESET}}$ is released.



NOTES:

(1) The bus status codes during these cycles depend on the instruction being performed at this time.

(2) $\overline{\text{RESET}}$ is sampled at every high-to-low CLKOUT transition. $\overline{\text{RESET}}$ is required to be active-low-for a minimum of three sample times so that the sequence can occur correctly.

(3) The reset context switch begins two CLKOUT cycles after $\overline{\text{RESET}}$ is sampled as having returned to the inactive-high level.

FIGURE 19 – RESET SEQUENCE

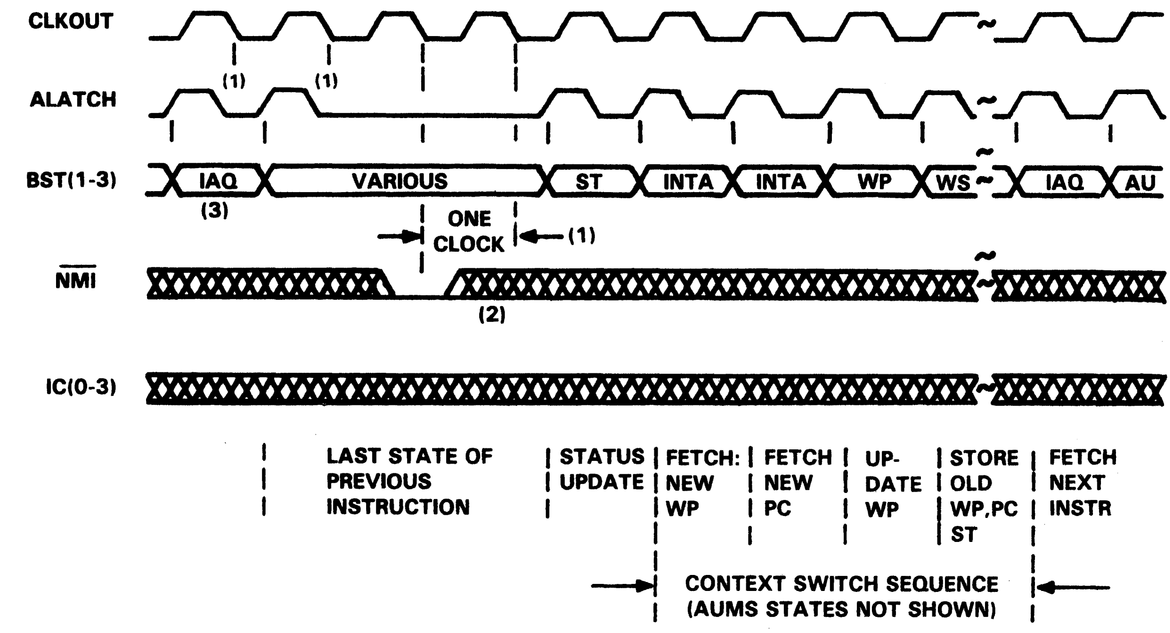
When $\overline{\text{RESET}}$ is released, a context switch to the level 0 service routine is initiated. The processor acquires the new WP and PC values from the trap vector located at memory address 0; it stores the old WP, PC and ST values in the new workspace; and it clears all status register bits and all internal error interrupt status bits to 0. If NMI is not active, the processor fetches the first instruction of the reset service routine. Otherwise, the NMI trap occurs after the context switch for the reset trap completes, but before the first instruction of the reset routine is executed.

A level 0 external interrupt is requested by pulling the processor's $\overline{\text{INTREQ}}$ input low while IC0-IC3 are all low. In general, the use of the level 0 interrupt requires that the request be removed when the INTA bus status code is output. Otherwise, the interrupt will be accepted a second time since it cannot be masked, and the return context will be lost. Note that the level 0 external interrupt is not the same as $\overline{\text{RESET}}$ but rather an external nonmaskable interrupt which uses the same trap vector as $\overline{\text{RESET}}$.

4.3 NON-MASKABLE INTERRUPT (NMI)

The NMI cannot be masked out. It is enabled by all values of the interrupt mask. The NMI implements ROM loaders, single-step and breakpoint functions for maintenance panels, and other user functions. An NMI request is generated by pulling the processor's $\overline{\text{NMI}}$ input low. This signal and its associated function are named "LOAD" in some previous 9900 family products.

An NMI request is handled according to the basic interrupt timing sequence described previously. The timing for the NMI is presented in Figure 20. As shown in Table 3, the NMI trap vector resides at memory address FFFC. The interrupt mask is automatically cleared when an NMI occurs.



NOTES:

(1) $\overline{\text{NMI}}$ is always sampled but will not be acknowledged until after the IAQ cycle.

(2) After an NMI context switch has been initiated, $\overline{\text{NMI}}$ can remain active low indefinitely without causing another NMI request to be generated. In order to generate another NMI request, $\overline{\text{NMI}}$ must be taken inactive high and be sampled at least once at the inactive level before being activated again. ($\overline{\text{NMI}}$ is sampled on the high-to-low transition of CLKOUT.)

(3) The prefetched instruction will be discarded as soon as the NMI request is recognized.

FIGURE 20 – NMI SEQUENCE

The processor almost always grants NMI request immediately following completion of the current instruction. The only exceptions to this statement are user-defined instructions emulated in Macrostore that use opcode >0384 to exit Macrostore (described in Section 7).

4.4 INTERRUPT LEVEL 2

The level 2 trap vector is used for external interrupts as well as for the following internal trap conditions: arithmetic fault, illegal instruction, and privileged opcode violation. Sampling of the level 2 external interrupt (and of all other interrupts as well) is delayed until the end of each instruction (Figure 18) to facilitate non-ambiguous error reporting. An error detected by external logic during execution of an instruction will be recognized (if level 2 interrupts are enabled) before the next instruction is executed. The PC value stored during the level 2 context switch points to the instruction following the one which generated the error. The external level 2 interrupt should be reserved for system errors such as memory faults and access violations. A memory error generated by the instruction prefetch should be delayed for one non-DMA bus cycle by external logic to insure that the memory error is associated with the prefetched instruction rather than with the instruction preceding it.

The error interrupt status bits are located at the I/O addresses shown in Table 4. Appropriate bits defined in Table 4 are set at the time the error occurs. A level 2 interrupt request is generated as long as any bit in the error status register, except the ILLOP (illegal instruction) flag, is set. The ILLOP flag is set under control of Macrostore emulation software, as described in Section 4.4.3. The error status bits can be examined by the level 2 interrupt service routine to determine the cause of the interrupt. The active bit(s) must be reset using a bit I/O operation (SBZ or LDCR) to acknowledge the condition and remove the interrupt request. This is not strictly necessary in the case of the ILLOP flag since it does not generate an interrupt request.

TABLE 4 – ERROR INTERRUPT STATUS BIT ASSIGNMENTS

	I/O ADDRESS	
ERROR FLAG	**INPUT BIT**	**OUTPUT BIT**
PRIVOP (privileged violation)	>1FDC	>1FDC
ILLOP (illegal instruction code)	>1FDA	>1FDA
AF (arithmetic overflow – ST4 and ST10 on)	>1FC8	>1FC8

Each bit is individually cleared by writing a 0 to the bit, e.g., by means of an SBZ or LDCR instruction. When I/O input operations are performed, the external IN input line is ignored. Writing a 1 to I/O address locations > IFC0 to > IFC7 will cause all three flags (PRIVOP, ILLOP and AF) to be set to 1. Writing a 1 to > IFD3 or > IFD4 will set the ILLOP or PRIVOP flag to 1, respectively.

When a level 2 interrupt has occurred, the level 2 service routine reads the I/O error interrupt status bits using the STCR or TB instruction to identify the source of the error. The interrupt is cleared by writing a 0 to the appropriate bit. The error interrupt status bits are automatically cleared by executing any of the following operations:

- RSET instruction
- Reset function (external device pulls $\overline{\text{RESET}}$ input low)
- I/O output operations to the bit(s) I/O address.

4.4.1 Arithmetic Fault Interrupt

The TMS99000 can be programmed to generate an interrupt when an overflow occurs during an arithmetic operation. This permits arithmetic fault checking to be performed without software overhead. The arithmetic fault interrupt is controlled using three programmable flags: bits 4 and 10 of the status register (Table 15) and AF of the error interrupt status bits (Table 4). ST4 is the arithmetic fault flag and is set to 1 whenever an overflow occurs. ST10 is the arithmetic fault enable bit and is set or cleared by the programmer to enable or disable, respectively, the arithmetic fault interrupt. AF of the error status register is automatically set when both ST4 and ST10 are 1. When set and the interrupt mask is greater than or equal to 2, AF generates a level 2 interrupt request, which is handled according to the standard interrupt sequence described in Section 4.2.

If an arithmetic overflow occurs while ST10 is 1 and the interrupt mask contains a value in the range 2 through 15, a level 2 interrupt occurs directly upon completion of the instruction causing the overflow. The PC value saved during the resulting context switch is the address of the first word of the instruction immediately following the instruction that caused the overflow.

The level 2 interrupt service routine must check ST4, ST10 of the saved status register in the routine's workspace register 15 (WR15) and AF to determine that the interrupt was caused by an overflow. The service routine, which is invoked by the arithmetic fault interrupt, must clear the AF bit and either bit 10 or bit 4 of WR15 before returning to the routine which caused the overflow. If this procedure is not followed, the arithmetic fault will falsely occur immediately upon the completion of the RTWP instruction.

Should the level 2 interrupt service routine be interrupted, in turn, by a higher-priority interrupt, the arithmetic fault condition is retained in AF until the program explicitly clears it. Similarly, ST4 and ST10 are preserved when the status register is saved during the context change; they are restored upon return from the higher-priority interrupt.

4.4.2 Macroinstruction Detection (MID) Trap

The MID trap permits the detection of illegal opcodes and the emulation of opcodes not defined in the processor instruction set. Emulation can be performed either in hardware or software. Hardware emulation takes place using the attached processor interface discussed in Section 8. Software emulation routines are contained in the Macrostore, described in Section 7.

The acquisition of an undefined opcode during an instruction fetch causes an MID trap to occur immediately before the processor attempts to execute the instruction. A list of the opcodes, which cause the MID trap, are listed in Section 10.5.19 and consists of all opcodes undefined in the basic instruction set of the processor. These are referred to as MID opcodes. The XOP instruction is also treated as an MID opcode in the event that bit 11 of the status register is set to 1.

Whenever the processor detects an MID opcode in the instruction stream, a check is made to determine whether an attached processor is prepared to execute the instruction.

If not, program control is transferred to the external Macrostore in the case of the TMS99105 or internal Macrostore in the case of the TMS99110 to allow the instruction to be emulated in software (see Section 4.4.4).

The sequence of actions that takes place during an MID trap is as follows. Upon detecting an MID opcode, the processor outputs an MID bus status code ($\overline{\text{MEM}}$ = 1, BST = 110). If an attached processor is prepared to execute the instruction, it responds to the MID status code by pulling the processor's $\overline{\text{APP}}$ input low. If $\overline{\text{APP}}$ remains inactive, program control is transferred to the Macrostore. The PC saved during the context switch points to the word following the MID opcode. If the MID opcode is followed by immediate data or address information, the emulation routine can use the saved PC value to access this information. In a likewise manner the program may use the saved workspace pointer (WP) to access operands in the calling routines workspace.

The MID trap is non-maskable and is higher in priority than any other interrupt except the reset function ($\overline{\text{RESET}}$ active low). An MID opcode always results in an MID trap regardless of the value of the interrupt mask. If an NMI request is received at the same time that an MID opcode is detected, the MID trap sequence will take place first and be followed immediately by the NMI sequence before the attached processor or Macrostore software begins to emulate the MID instruction. This permits the NMI input signal to control single-stepping in conjunction with MID opcodes and microcoded processor opcodes alike.

When a user program attempts to execute an MID opcode that is defined neither by an attached processor nor by a Macrostore emulation routine, the Macrostore software should detect this fact and initiate a level 2 interrupt. This is accomplished by the special form of the RTWP instruction (opcode >0382), which causes the processor to set the ILLOP bit of the error status bits and then exit Macrostore memory space. This provides an indicator to the level 2 trap routine undefined opcode violation. If Macrostore is disabled at Reset, then an MID opcode will automatically cause the ILLOP bit to be set and a level 2 trap to occur.

4.4.3 Illegal Instruction (ILLOP) Interrupt

The illegal instruction interrupt is a result of a Macrostore exit through the execution of the special form of the RTWP instruction (opcode >0382) when in Macrostore (section 4.4.2). The level 2 interrupt routine can identify the illegal instruction interrupt by interrogating the ILLOP bit of the error status bits.

This method is not reliable in detecting illegal instructions within level 0 or level 1 interrupt routines. The context linkage may be lost if two successive level 0 or level 1 external interrupts occur unless a software stack is created.

The ILLOP trap permits the system to respond to illegal opcodes. When an illegal opcode is first encountered, the processor performs two preliminary checks before setting the ILLOP error interrupt status flag and vectoring to the level 2 interrupt routine. First, the processor outputs the MID (macro-instruction detected) bus status code to determine whether an attached processor is prepared to execute the instruction. If not, the processor transfers control to the emulation software in Macrostore to determine whether it recognizes the opcode. If this test also fails, the opcode is illegal. The Macrostore software sets the ILLOP flag and returns control to the user's program in main memory. The processor traps immediately to the level 2 interrupt routine before it has a chance to resume execution of the user's program. Once the trap has occurred, the level 2 interrupt routine checks the ILLOP flag to determine if the trap was caused by an illegal instruction. The ILLOP interrupt is non-maskable.

For further information, refer to the discussion of MID opcodes in Section 4.4.2.

4.4.4 Privileged Opcode Violation Interrupt

When a privileged opcode violation occurs, as described in Section 6, PRIVOP, the error interrupt status bit is set and, if the interrupt mask is greater than or equal to 2, generates a level 2 interrupt request. The offending instruction is permitted to complete any operand fetches it requires, but following detection of the violation, all further attempts to write to the I/O address space are inhibited. As soon as the offending instruction completes execution, a level 2 interrupt takes place unless the interrupt mask value is 0 or 1. The trap sequence for the level 2 interrupt consists of a standard context switch, after which program control is transferred to the level 2 service routine. The routine checks the PRIVOP bit to determine if the interrupt was caused by a privileged opcode violation. PRIVOP should be cleared by the service routine before returning; otherwise, another level 2 interrupt will occur upon return unless the interrupt mask is set to a value of 0 or 1.

If a privileged opcode violation occurs while the interrupt mask is 0 or 1, the level 2 interrupt is disabled and does not take place. The PRIVOP bit is still set, and any write operations attempted by the offending instruction are inhibited as before. Execution of succeeding instructions will occur normally, however, until the interrupt mask is set to a value of 2 or greater, at which point the PRIVOP bit, which has remained set, causes a level 2 interrupt to occur.

4.4.5 Applicable Bus Status Codes

The INTA (interrupt acknowledge) bus status code is output by the processor to indicate that a reset, NMI, XOP (if ST11 =0), external interrupt or any of the level 2 interrupts is in progress. The INTA is output during the fetch of the WP and PC values from the trap vector for the interrupt or XOP.

If desired, the INTA bus status code can be used as a signal to automatically acknowledge an interrupting device. The acknowledgment signal is generated by hardware external to the processor that can recognize the INTA code and determine the interrupt level by decoding address bits A10 through A13.

5. TMS99000 INPUT/OUTPUT INTERFACE

5.1 DESCRIPTION

The TMS99000 provides both bit-serial and bit-parallel I/O to meet the requirements of both bit, byte and word peripheral applications. TMS99000's I/O is a command-driven direct I/O interface that supports bit, byte and word data transfers. The I/O address space contains 32768 peripheral input locations, and 32768 output locations. The first 16384 I/O addresses (input or output) are bit locations accessed in bit-serial fashion; the last 16384 I/O addresses are word or byte locations (as specified by the user) accessed in bit-parallel fashion. Figure 21 provides the I/O address map for the TMS99000.

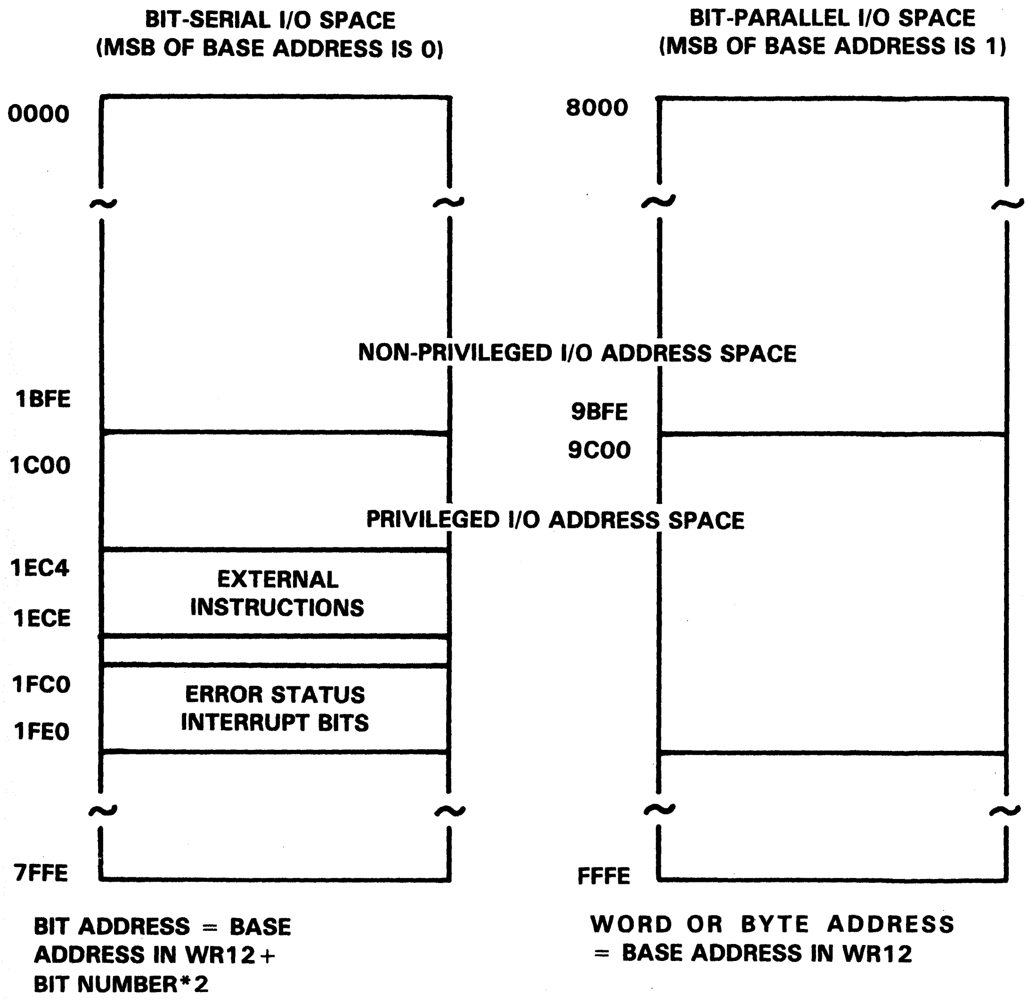


FIGURE 21 – I/O ADDRESS MAP

The I/O interface utilizes the same signal lines as the interfaces for main memory and external Macrostore, as indicated in Figure 22. The I/O address space, however, is logically distinct from the memory and macrostore address spaces. The timing for I/O read and write operations is presented in Figures 23 and 24. I/O operations are accompanied by the I/O bus status code ($\overline{\text{MEM}}$ = 1, BST = 011) to distinguish them from accesses of memory or external Macrostore. (In fact, only two signals, $\overline{\text{MEM}}$ and BST2, are required for this purpose; see Table 2.)

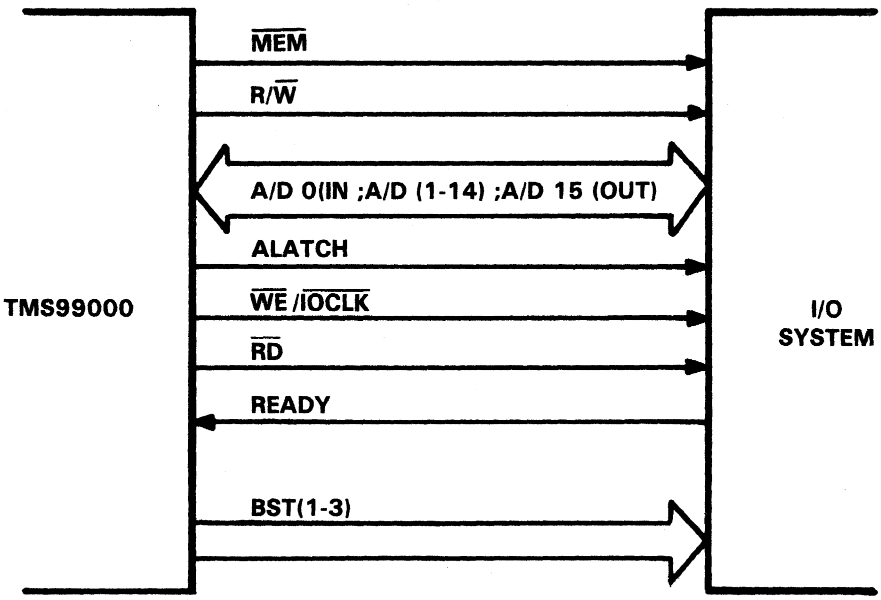
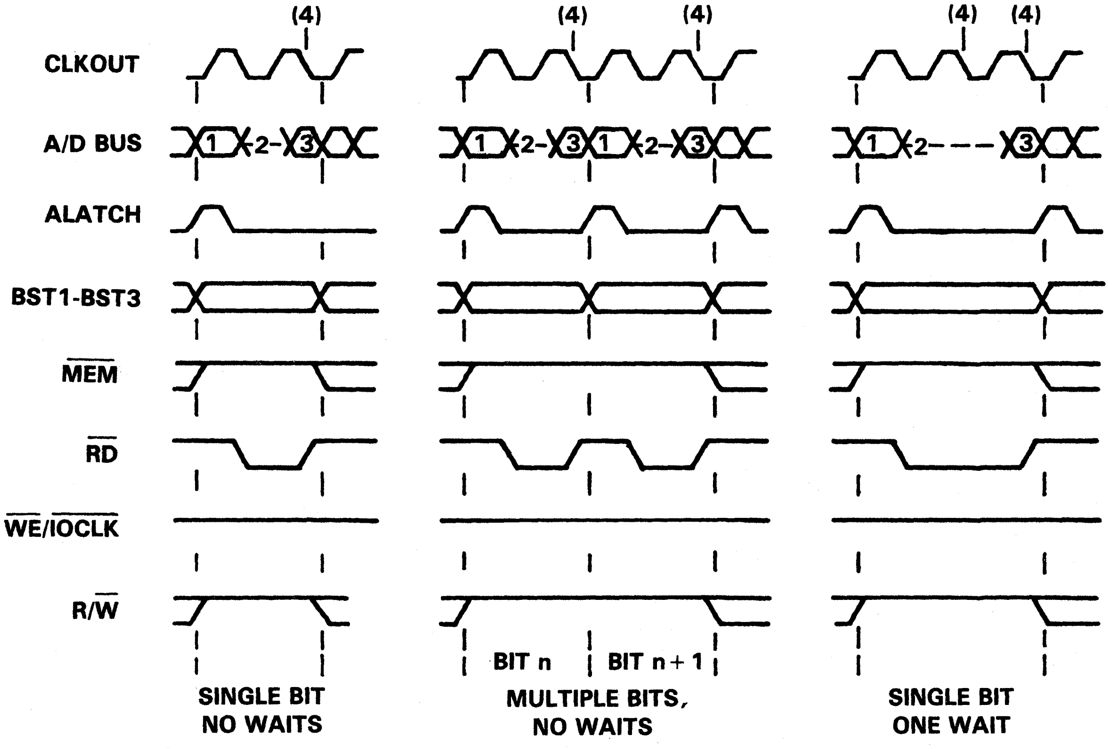


FIGURE 22 – TMS99000 I/O INTERFACE



NOTES:

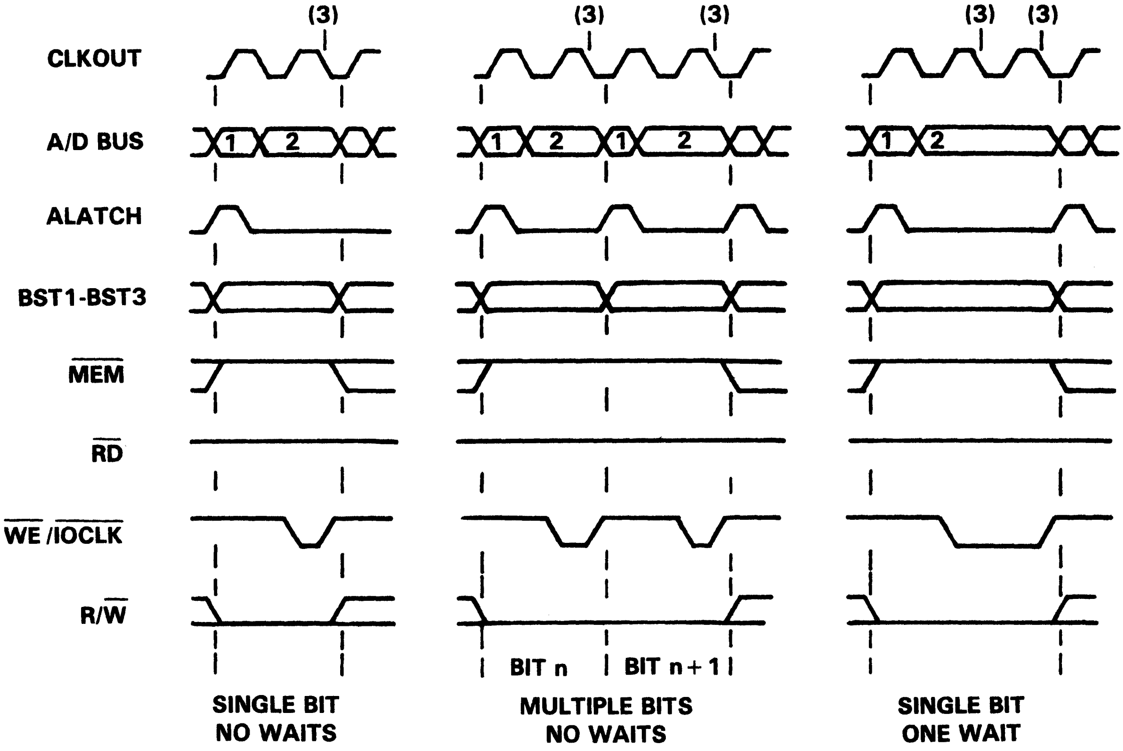
(1) Valid address, PSEL high (internal ST8 = 0)

(2) Bus in input mode (drivers are tristated)

(3) If MSB of address is 0, I/O bit must be valid on D0/I/IN. If MSB of address is 1, I/O word must be valid on A/D(0-15), and I/O byte on A/D(0-7).

(4) READY is sampled at this time.

FIGURE 23 – TMS99000 I/O TIMING – INPUT OPERATION



NOTES:

(1) Address is valid, and $\overline{\text{PSEL}}$ is high (internal ST8 = 0).

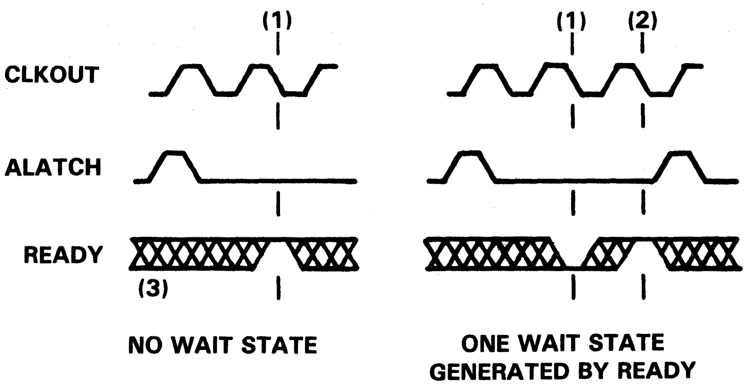
(2) If MSB of address is 0, then valid I/O bit is on D15/OUT. If MSB of address is 1, then valid I/O word is on A/D(0-15), and I/O byte is on A/D(0-7).

(3) READY is sampled at this time.

FIGURE 24 – TMS99000 I/O TIMING – OUTPUT OPERATION

Each I/O cycle begins with an ALATCH pulse, the falling edge of which latches externally the 15 address bits A0 to A14 and the $\overline{\text{PSEL}}$ page select signal. If A0, the MSB of the address, is 0, a bit-serial I/O operation is performed; if A0 is 1 during an LDCR or STCR instruction, a bit-parallel (byte or word) I/O operation is performed. The $\overline{\text{PSEL}}$ output signal is forced high during all I/O cycles regardless of the actual state of ST8 of the processor's status register (Section 2.3.2). Following the ALATCH pulse, data is input or output on the address-data lines and R/$\overline{\text{W}}$ is taken high or low to indicate whether an input or output operation is to be performed. Serial I/O accesses utilize the A0/D0/IN line for reads, and the $\overline{\text{PSEL}}$/D15/OUT line for writes. Parallel I/O operations utilize all 16 data lines (D0-D15) for word transfers, and the first eight (D0-D7) for byte transfers. I/O write operations are accompanied by a low pulse on the $\overline{\text{WE}}$/$\overline{\text{IOCLK}}$ output; I/O read operations are accompanied by a low pulse on $\overline{\text{RD}}$.

The minimum-length I/O cycle is two machine states (two CLKOUT periods) in duration. If, during the second machine state of a I/O cycle, READY is low, the cycle is extended by one wait-state. Holding READY low generates additional wait-states until READY is taken high prior to the high-to-low transition of CLKOUT. Figure 25 illustrates the relationship between I/O wait states and the READY line.



NOTES:

(1) First sample time of READY in I/O cycle.

(2) Second sample time of READY in I/O cycle. Additional wait states are generated by keeping READY low at this and subsequent sample times.

(3) XXXXXX denotes don't care.

FIGURE 25 – WAIT-STATE GENERATION FOR I/O CYCLES

The TMS99000 instruction set contains five I/O-oriented instructions. Three of the I/O instructions are used to perform single-bit operations in the first 16384 bits of the I/O address space. (See Figure 22.) These are the TB (test bit), SBO (set bit to one) and SBZ (set bit to zero) instructions. The remaining two I/O instructions perform multiple-bit operations in either bit-serial or bit-parallel fashion, depending on which half of the I/O space is being addressed.

5.2 SINGLE-BIT I/O OPERATIONS

The single-bit instructions facilitate the testing and/or modification of a particular bit in a device. The three single-bit I/O instructions, TB, SBO and SBZ, are executed as follows. The TB instruction reads the bit from the addressed I/O location onto the A0/D0/IN line, and this bit is placed in status register bit 2 (EQ). The SBO instruction outputs a one on the $\overline{\text{PSEL}}$/D15/OUT line, and the SBZ outputs a zero on this line.

The processor develops the address for a single-bit I/O operation from the base address contained in bits 0 to 14 of WR12 and from the signed displacement field contained in bits 8 to 15 of the instruction. As indicated in Figure 26, the signed displacement *2 is added to the base address to generate the effective I/O address output onto the bus. The displacement allows two's complement addressing from base – 128 through base + 127. Note that for single-bit I/O instructions, SBO, SBZ, and TB, the most-significant bit of WR12 does not affect the operation (i.e., no parallel operations).

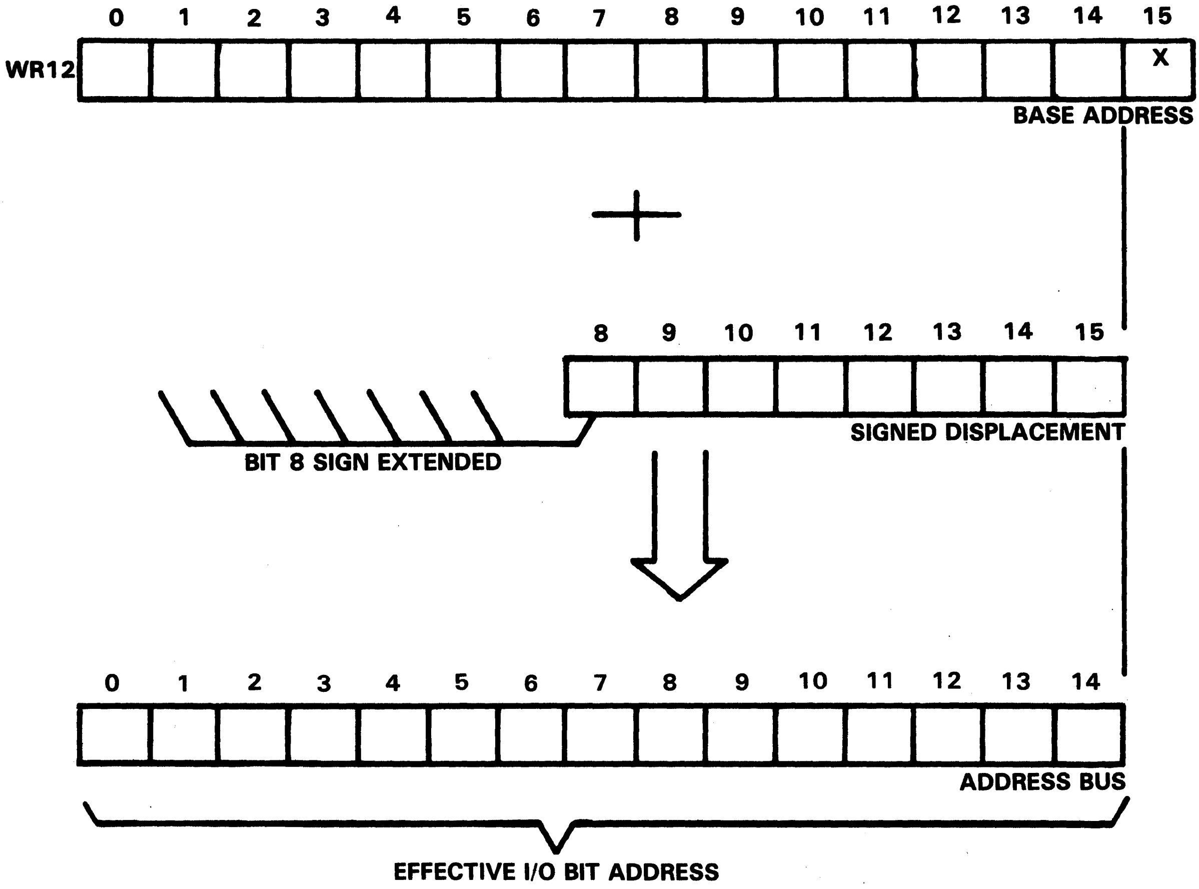


FIGURE 26 – SINGLE-BIT I/O ADDRESS DEVELOPMENT

5.3 MULTIPLE-BIT SERIAL I/O OPERATIONS

The STCR and LDCR instructions specify multiple-bit I/O operations. The starting address in I/O address space is loaded into WR12 prior to executing STCR or LDCR. When the MSB (bit 0) of the address in WR12 is 0, the transfer is performed in bit-serial fashion rather than in parallel. During a multiple-bit, bit-serial I/O transfer, the first bit is read from or written to the address pointed to by bits 0 through 14 of WR12. Consecutive bits are read from or written to I/O locations separated by an address increment of + 1. The contents of WR12 are not altered by execution of the serial STCR or LDCR instructions.

A multiple-bit serial I/O transfer is represented in Figure 27. Although a full 16-bit transfer is indicated in the figure, any number of bits from one to 16 can be specified. The transfer mechanism results in an order reversal of the bits; that is, bit 15 of the memory word (or bit 7 of the byte) corresponds to the bit stored at the lowest I/O address, and bit 0 corresponds to the bit stored at the highest I/O address.

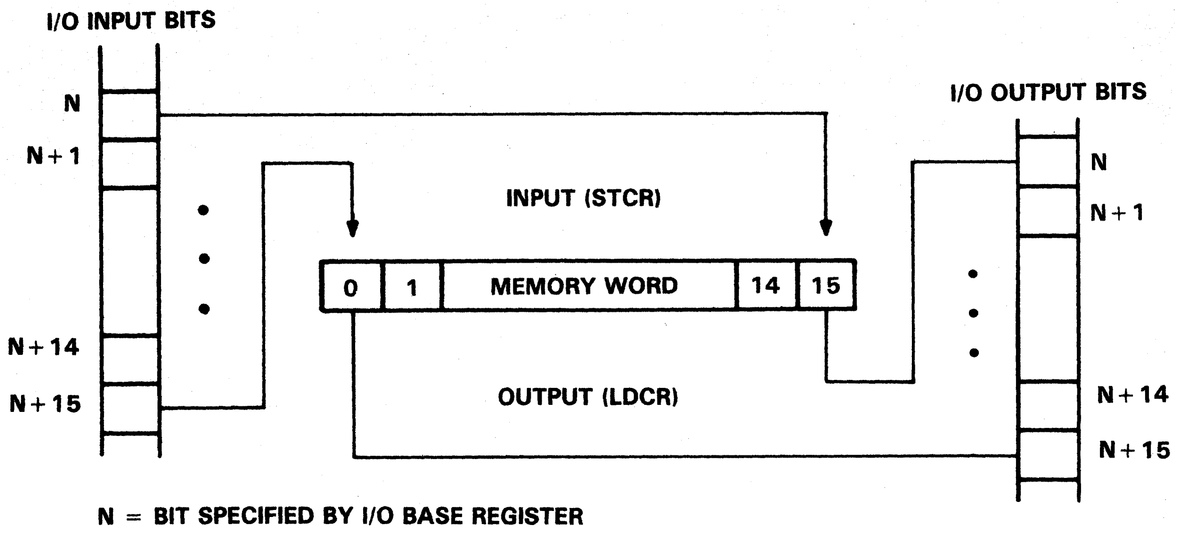


FIGURE 27 – LDCR/STCR DATA TRANSFERS

The first word of an STCR or LDCR instruction contains a 4-bit CNT (count) field, which specifies the number of bits to be transferred. If CNT is 0, then 16 bits are transferred. If CNT is in the range 1 to 8, the effective source operand address from the STCR or LDCR instruction is treated as a byte address; otherwise, it is treated as a word address.

The LDCR instruction reads a word (or byte) from the memory and writes all or part of it to the I/O in bit serial fashion. Beginning with the rightmost bit in the word (or byte) and moving from right to left, each consecutive bit is output through the I/O interface until the specified number of bits has been transferred.

The STCR instruction reads data from the I/O and transfers it to memory. If the specified number of bits to be transferred from the I/O is less than 9, they are stored right-justified in the addressed memory byte, and the leading bits are cleared to 0. If the operation involves from 9 to 16 bits, the data is stored right-justified in the addressed memory word, and the leading bits cleared to 0. When the instruction is completed, the bit from the lowest I/O address occupies the LSB position in the memory word or byte.

5.4 PARALLEL I/O OPERATIONS

When the MSB (bit 0) of the I/O base address in WR12 is 1, the multiple-bit I/O transfer specified by an STCR or LDCR instruction takes place in a single I/O transfer, i.e., in parallel. Either a word or byte is transferred as determined by the 4-bit CNT field in the first word of the LDCR or STCR instruction. If the CNT field is within the range of 9 to 15, then a word is transferred. If the CNT field is within the range 1 to 8, then a byte is transferred. For parallel I/O, CNT is restricted to (binary) 0010, 0011, 1010, and 1011. The CNT field selects the mode of transfer as shown below:

if CNT = 0010, then byte transfer
if CNT = 0011, then byte transfer with WR12 auto increment
if CNT = 1010, then word transfer
if CNT = 1011, then word transfer with WR12 auto increment.

The automatic increment of WR12 is provided to facilitate block transfers of data to and from devices in the parallel I/O address space.

5.5 APPLICABLE BUS STATUS CODES

I/O cycles are identified by the I/O bus status code (Table 2).

5.6 EXTERNAL INSTRUCTIONS

The TMS99000 has five external instructions that allow user-defined off-chip functions to be initiated under program control. These instructions are CKON, CKOF, RSET, IDLE and LREX. These names are arbitrary. The user may define the external function performed by these instructions.

Execution of CKON, CKOF, RSET or LREX causes a bit value of 0 to be written to one of the I/O addresses specified in Table 5. Following the single I/O write cycle, execution proceeds to the next instruction. RSET is the only external instruction that can affect the ST (status) register or the error interrupt status bits. In privileged mode (ST7 = 0), RSET causes ST9-ST15 and the AF, ILLOP, and PRIVOP bits of the error status bits to be cleared. This is followed by a status update cycle (ST bus status code) to notify external devices of the change in status. In user mode (ST7 = 1), the ST and error status bits are unaffected, and RSET is similar in effect to CKON, CKOF and LREX.

TABLE 5 – EXTERNAL INSTRUCTION CODES

INSTRUCTION	I/O BASE ADDRESS
IDLE	1EC4
RSET	1EC6
CKOF	1ECC
CKON	1ECA
LREX	1ECE

IDLE differs from the other external instructions in that its function is predefined. Execution of IDLE causes the processor to enter and remain in the idle state until a $\overline{\text{RESET}}$, $\overline{\text{NMI}}$ or unmasked external interrupt occurs. While in the idle state, a bit value of 0 is written repeatedly to I/O address >1EC4 (i.e., the $\overline{\text{WE/IOCLK}}$ output is pulsed continually). Upon leaving the idle state, a context switch takes place to service the interrupt. The PC value saved during the context switch points to the address of instruction following the IDLE instruction.

The timing for the I/O write operation, or operations in the case of IDLE, follows that given in Figure 24. Each I/O write cycle is accompanied by the I/O bus status code (Table 2).

When the processor receives a hold request ($\overline{\text{HOLD}}$ low) while in the idle state, the processor enters the hold state directly from the idle state. It reenters the idle state as soon as the hold request is deactivated.

6. PRIVILEGED MODE

For hardwired system protection in a user/supervisor programming environment, certain instructions performing I/O and control functions are designated as "privileged". When the system is placed in the user or "non-privileged" mode, any attempt to execute one of these instructions will result in abortion of the instruction and an interrupt request through the level 2 trap vector. (See Section 4.)

The system can be placed in the user mode by setting status bit 7 to 1 by means of an LST or RTWP instruction. The system is placed in the privileged mode by the occurrence of any interrupt, execution of the XOP instruction, by the assertion of the $\overline{\text{NMI}}$ or $\overline{\text{RESET}}$ input signals, or during Macrostore operations.

When a privileged opcode violation is detected, error status bit PRIVOP is set, and this, in turn, generates a request for a level 2 interrupt as described in Section 4.4.3. The following instructions are privileged: CKON, CKOF, IDLE, LIMI, LREX and RSET. The use of the following instructions is qualified in user mode: LDCR (I/O), RTWP, SBO (I/O), SBZ (I/O), and LST. In processors with LDD and LDS instructions implemented in Macrostore such as the TMS99110, these instructions are privileged. Section 10.5 should be consulted to determine the restrictions placed on each of these instructions in user mode.

The LDCR instruction is a privileged instruction for byte and word transfers to output addresses falling within the range specified by Figure 21. Similarly, the SBO, SBZ I/O instructions are privileged for bit I/O operations falling within the same range.

The LST operation is dependent upon whether the processor is in the privileged or non-privileged mode when the instruction is executed. While in the privileged mode (ST7 = 0), the LST instruction modifies all 16 bits of the status register. While in user mode (ST7 = 1), only bits 0 through 5 and bit 10 of the workspace register specified in the W field are placed in the status register, and loading these bits has the side effect of clearing ST6. Similarly, return workspace pointer (RTWP) instruction will cause all bits of the status register to be replaced when in privileged mode and only the seven bits discussed when in a non-privileged mode.

Section 10.5 discusses the operation of these instructions in more detail.

7. MACROSTORE INTERFACE AND OPERATION

7.1 DESCRIPTION

Macrostore is a special feature of the TMS99000 that permits new instructions to be defined and emulated in a manner completely transparent to programs residing in main memory. It provides the capability for adding new functions and enhancing the performance of specific kernels of software, thereby increasing the total system performance. Macrostore permits software kernels to be encapsulated within the TMS99000 system in a manner that makes them virtually indistinguishable in operation from functions implemented in hardware. This is accomplished by providing a 64K byte address space that is logically distinct from the main memory and I/O address spaces. Macrostore functions as a control store for the TMS99000 but is programmed in assembly language rather than microcode. Internal to the TMS99000 are 1024 bytes of Macrostore ROM (MROM) and 32 bytes of Macrostore RAM (MRAM). The access time of the on-chip Macrostore is one machine state. Emulation routines in the internal Macrostore execute at the full speed of the processor since no wait states are required to access the on-chip MROM and MRAM. While executing in the Macrostore, certain control capabilities are provided that are not available to programs executing in the main memory.

7.2 THE MACROSTORE INTERFACE

7.2.I Timing

The timing signals generated during accesses of external Macrostore are identical to the memory timing described in Sections 3.2.1 and 3.2.2, with the following exceptions. The only bus status codes (Table 2) output are the AUMS and AUMSL codes. (AUMSL is output if an ABS, TSMB or TCMB instruction is executed in Macrostore. Otherwise, AUMS is output.)

Another difference between Macrostore accesses versus main memory accesses is the operation of the $\overline{\text{PSEL}}$ output. In main memory accesses, the $\overline{\text{PSEL}}$ output represents the inverted state of the ST8 bit of the status register unless a long distance source/long distance destination instruction (LDS, LDD) is in effect (see Section B.3). (The LDS and LDD instructions apply to the TMS99110 only; see Appendix B.) If a LDS or LDD instruction is in effect per the description in Appendix B, the $\overline{\text{PSEL}}$ output will represent the logic state of the ST8 bit without inversion. A complete description of the LDS and LDD instructions is given in Appendix B, Section B.3. For Macrostore accesses, the $\overline{\text{PSEL}}$ output is not guaranteed; thus it should not be used for paging Macrostore memory.

The AUMS and AUMSL bus status codes differentiate between external Macrostore accesses and memory and I/O accesses.

7.2.2 Wait States

Accesses of on-chip Macrostore require only a single machine state to complete. If the Macrostore is extended using an external RAM or ROM that is too slow to respond in a single machine state, external control logic must cause wait-states to be generated by pulling the 99000's READY input low until the access is ready to complete. The generation of wait-states is identical to main memory wait state generation described in Section 3.2.5.

7.2.3 Organization

The internal Macrostore consists of 1024 bytes of MROM and 32 bytes of MRAM. The MROM resides at addresses >0800 to >0BFE. The MRAM resides at addresses >0000 to >001E and serves as workspace storage during Macrostore execution. External Macrostore may be added in the form of off-chip ROM or RAM residing at addresses in the range >1000 to >FFFE. A map of the Macrostore address space appears in Figure 28.

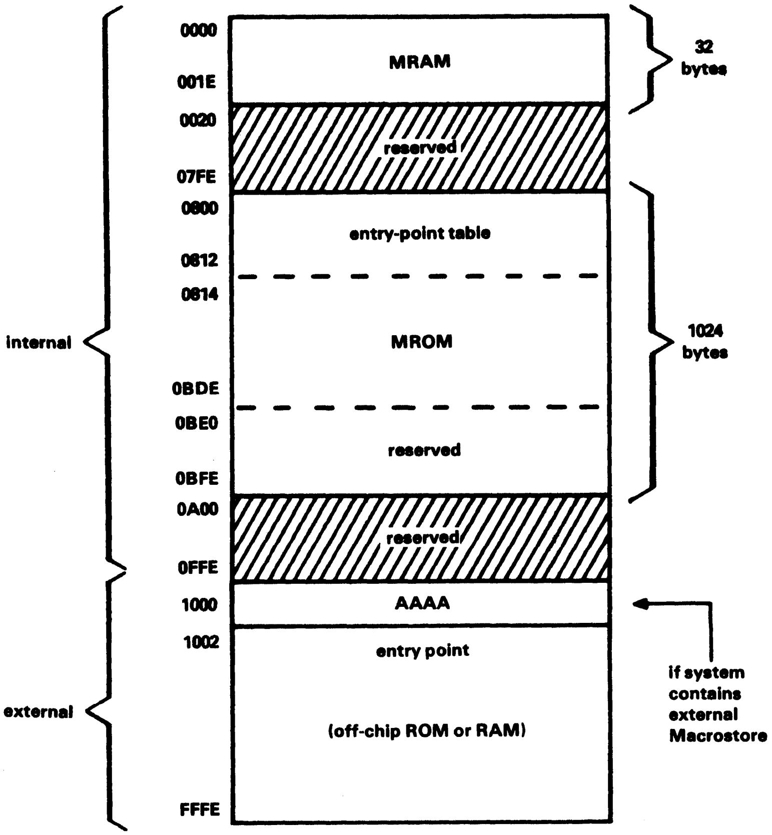


FIGURE 28—ADDRESS MAP OF MACROSTORE

7.2.4 Modes of Operation

The TMS99000 operates in one of three modes which determine the operation of Macrostore. These modes are summarized in Table 6 and in the following paragraphs.

TABLE 6 – MACROSTORE OPERATING MODES

MODE	EFFECT	ENTRY PROCEDURE
Standard	On-chip ROM (1K bytes) and RAM (32 bytes) is assumed. External Macrostore memory expansion from >1000 through >FFFE.	$\overline{\text{APP}}$ pin is a high level at reset.
Prototyping	On-chip ROM address range (>0800 – >0FFE) is mapped off-chip for use of external Macrostore memory. On-chip RAM is available.	$\overline{\text{APP}}$ pin is taken low when $\overline{\text{RESET}}$ is pulled low and is released when $\overline{\text{RESET}}$ is released.
Baseline	All Macrostore memory space is disabled and the attached processor interface is disabled.	$\overline{\text{APP}}$ pin is tied to ground.*

*If $\overline{\text{APP}}$ is brought high anytime after $\overline{\text{RESET}}$, the processor will enter the prototyping mode.

7.2.4.1 *Standard Mode*

In standard mode, the on-chip MROM and MRAM are both enabled, permitting the firmware contained in the MROM to be utilized. During accesses of on-chip MROM and MRAM, the AUMS and AUMSL status codes are output, and the WE/IOCLK and RD outputs both remain inactive high.

While executing in Macrostore, a read or write to a Macrostore address in the range >1000 to >FFFE results in an access of external Macrostore. During this access, either the RD or WE/IOCLK output goes active low, depending on whether the Macrostore location is being read from or written to. The timing for the access is the same as that described for an access performed by a program residing in main memory, as described in Section 3.2, with the exception that the only bus status codes output are AUMS and AUMSL. This is consistent with the treatment of Macrostore execution as a special type of internal operation. The AUMS and AUMSL bus status codes are used by external decode logic to distinguish accesses of external Macrostore from accesses of main memory and I/O locations. Accesses of external Macrostore are, in turn, distinguished from other kinds of internal operations by observing the RD and WE/IOCLK outputs, which are active during Macrostore accesses, but not during other types of internal operations.

The TMS99000 is placed in standard mode by keeping the APP input high while RESET is pulled low at system initialization.

7.2.4.2 *Prototyping Mode*

In prototyping mode, the TMS99000's internal MROM is disabled, but the MRAM remains enabled. A read or write to a Macrostore address in the range >0000 to >001E results in an access of the on-chip MRAM, but a read from or write to any Macrostore address in the range >0800 to >FFFE results in an external Macrostore access. As in the standard mode, the WE/IOCLK and RD outputs are active only when the Macrostore read or write is off-chip. The AUMS and AUMSL bus status codes are output during accesses of both internal and external Macrostore.

The processor is placed in prototyping mode by pulling the RESET and APP inputs low together during system initialization and releasing them at the same time. In systems without attached processors, the RESET and APP pins can simply be tied together.

One use of prototyping mode is to permit external RAM or ROM occupying Macrostore addresses >0800 to >0BFE to emulate on-chip MROM during development and testing of Macrostore software.

7.2.4.3 *Baseline Mode*

In baseline mode all Macrostore memory space is disabled. In the event a MID opcode is encountered, the TMS99000 will cause a level 2 interrupt to occur and the ILLOP bit of the error status register (Section 4.4) will be set. The level 2 interrupt routine then may emulate the opcode or the opcode may be handled as an illegal opcode violation. In baseline mode the attached processor interface is also disabled. Thus the APP input pin will not be tested on the occurrence of a MID opcode. The level 2 interrupt will be implemented immediately.

The TMS99000 is placed in baseline mode by pulling the APP input low at reset. It remains in baseline mode as long as AAP remains low. Typically, this is accomplished simply by tying APP to ground. (Note that if APP goes high after RESET, the processor will enter the prototyping mode.)

7.3 MACROSTORE CAPABILITIES

7.3.1 Entry Procedure

When the TMS99000 is executing a program residing in main memory and a MID opcode is encountered, the APP pin is tested to determine whether an attached processor is prepared to respond to the MID opcode. If not, program control is transferred to the Macrostore. A MID opcode is an undefined opcode in the basic TMS99000 instruction set, or an XOP executed while ST11 = 1.

The Macrostore is entered via an entry point table occupying the first ten words of the MROM, shown in Table 7. Each entry in the table contains the start address in MROM of an emulation routine for a particular group of MID opcodes. When a MID opcode is encountered in the program in main memory, instruction execution transfers to the MROM address in the entry-point table corresponding to that opcode. Undefined single-word opcodes are divided into eight groups with the entry addresses for each group as indicated in Table 7. Undefined two-word opcodes are treated as a 9th group, and XOPs, when ST11 = 1, as a 10th.

TABLE 7 – MACROSTORE ENTRY VECTORS

TABLE LOCATION	MID† OPCODES
0800*	0000-001B, 001E-0028, 002B-007F, 00A0-00AF, 00C0-00FF
0802*	0100-013F
0804*	0210-021F, 0230-023F, 0250-025F, 0270-027F, 0290-029F, 02B0-02BF, 02D0-02DF, 02E1-02FF
0806*	0301-031F, 0320-033F, 0341-035F, 0361-037F, 0381-039F, 03A1-03BF, 0EC1-03DF, 03E1-03FF
080A*	0C00-0C08, 0C0C-0CFF
080A*	0D00-0DFF
080C*	0E00-0EFF
080E*	0F00-0FFF, 0780-07FF
0810	AM, SM, SRAM, SLAM, TMB, TCMB, TSMB (if the second word is illegal)
0812	XOP (if ST11 = 1)

*Bits 5, 6 and 7 of the MID Opcode select one of eight entry-table locations.

†The opcodes reserved for the LDD and LDS instructions should not be used as MID opcodes.

A context switch occurs after the entry-point address has been read from the table. The workspace pointer is set to 0000 and the program counter is set to the address from the entry-point table. The old WP, PC, and status are saved in the MRAM locations corresponding to WR13, WR14 and WR15, respectively. The PC value saved in WR14 always points to the word immediately following the MID opcode. If a two-word MID opcode was encountered, the PC value always points to the word immediately following the first word of the two-word opcode.

Prior to transferring program control to the Macrostore emulation software, the MID opcode responsible for causing the MID trap is automatically placed in registers 3 and 5 of the Macrostore workspace. If the first word of an instruction causes the MID trap, the (entire) first word is placed in WR5. If the second word of an instruction causes the MID trap, the (entire) second word of the instruction is placed in WR5, and bits 10, 11, 14, and 15 of the first word of the instruction are placed into bits 10, 11, 14, and 15 of WR3. In the latter case, bits 10, 11, 14, and 15 are sufficient to uniquely identify the possible first word of an opcode in which the second word is illegal. The identification is performed as follows: Table 8 enumerates all the 2-word opcodes in the TMS99000 instruction set. These instructions are divided into 3 groups. Bits 10 and 11 identify the group. Each group contains 2 or 3 opcodes. Bits 14 and 15 serve to identify the individual opcodes within each group.

WR3 IN MACROSTORE:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LD FLAGS			0	0	0	0	0	0	0	GROUP		0	0	OP#	

LONG-DISTANCE
FLAGS

BITS 10, 11, 12 & 15
ENCODE FIRST WORD
OF TWO-WORD OPCODE

Bits 0, 1 and 2 of WR3 are initialized to the value of the 99110's long-distance flags upon entry to Macrostore. These flags indicate whether an LDS or LDD instruction is currently in effect, as explained in Section 10.

TABLE 8—INSTRUCTIONS WITH TWO-WORD OPCODES

MNEMONIC		FIRST INSTRUCTION WORD															
Group 1:	TMB	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1
	TCMB	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0
	TSMB	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	1
Group 2:	AM	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
	SM	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1
Group 3:	SLAM	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
	SRAM	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0

Bits 10 and 11 identify group

Bits 14 and 15 identify opcode within group

7.3.2 Exit Procedure

Macrostore is generally exited by executing a RTWP instruction (opcode >0380). Interrupts are sampled prior to executing the next instruction. In those instances where interrupts (maskable or non-maskable) should not be sampled before executing the next instruction, the exit from Macrostore is invoked using the opcode >0384, a special form of the RTWP instruction. In either case, the WP, PC, and ST registers are updated with WR13, WR14, and WR15 from the MRAM. The >0384 exit ties the Macrostore operation to the execution of the instruction that follows the MID instruction. For example, it is used in emulating the LDD and LDS instructions, described in Section 10.

If the Macrostore is entered upon detection of a MID opcode, and the emulation software in Macrostore determines that it does not recognize the MID opcode as valid, the software must transfer control to the level-2 interrupt service routine, which has the responsibility for dealing with illegal opcodes. The emulation software in Macrostore uses the opcode >0382, another special form of the RTWP instruction, to exit the Macrostore under these conditions. When this opcode is executed in Macrostore, the processor sets the ILLOP flag of the error status bits (Section 4.4) before executing the RTWP operation. Consequently, following the context switch back to the program in main memory that contains the undefined opcode, the ILLOP flag forces a trap to the level-2 interrupt service routine. The ILLOP interrupt is non-maskable and cannot be disabled by the interrupt mask in ST12-ST15.

The opcodes >0380, >0382 and >0384 provide the only means for performing an exit from Macrostore. To perform an RTWP in Macrostore (i.e., inverse of a BLWP) the opcode >0381 should be used. This opcode will allow for return branching in the Macrostore address space without exiting Macrostore.

7.3.3 Macrostore Execution

During Macrostore execution, several processor functions are modified to provide increased control. These are described below.

7.3.3.1 Status Register

The contents of the status register are not affected by the context switch to Macrostore that follows detection of a MID opcode. Macrostore routines are "super-privileged," meaning that they can alter the contents of the status register and perform other privileged operations regardless of the value of the privileged mode flag, ST7. An ST bus status code (Table 2) is output from the processor when a Macrostore routine alters the ST register by means of the LST instruction. When the status register of the original main memory program environment must be modified, the appropriate bits of WR15 must be modified prior to Macrostore exit.

During emulation of an MID opcode in Macrostore, the emulation routine can modify the ST register value saved in WR15 in accordance with the results. During the context switch that follows the exit from Macrostore, the new status is loaded into the ST register. If the status value saved in WR15 has not been changed since the entry to Macrostore, it will be restored in its original form.

While in Macrostore, the setting of both ST4 and ST10 to 1 does not cause the AF flag of the error status register to be set to 1. If an arithmetic fault interrupt is to be generated, then bit 4 and 10 of WR15 should be set so that an arithmetic fault trap will occur when the context switch out of Macrostore is made.

If it is required to modify the status bits of the main memory routine's status register prior to context switching out of Macrostore, the appropriate bits of WR15 should be modified. (Note if ST7 = 1, then status bits 6 to 15 will be set according to Section 6.)

7.3.3.2 *Interrupts*

All interrupts except $\overline{\text{RESET}}$ are inhibited while executing from Macrostore. However, pending interrupts can be detected using the conditional jumps described in Section 7.2.7.5.

7.3.3.3 *Macrostore Workspace Registers*

When Macrostore is initially entered, the Workspace Pointer is set to zero so that the internal Macrostore RAM is utilized as the workspace. The Workspace Pointer may be set to another value by the LWP, LWPI, or BLWP instruction when it is desired that the workspace be located in external Macrostore RAM.

The workspace registers located in internal RAM have special uses associated with the evaluate address (EVAD) instruction described in Section 7.3.3.5. For this reason care must be exercised in assuring that the EVAD instruction is used only with the Workspace Pointer equal to zero.

Table 9 lists the dedicated functions of the workspace registers when the workspace pointer bits 11 to 15 are equal to zero as is the case when Macrostore is first entered. Table 10 lists the bus status codes of the workspace registers when the workspace pointer bits 11 to 15 are not equal to zero (i.e., external RAM is used).

TABLE 9 – DEDICATED MRAM REGISTER FUNCTIONS
[WP bits 11 to 15 are all zero]

REGISTER	MAIN MEMORY ACCESS	BUS STATUS CODE	MOD BY EVAD	MOD BY MACRO	USAGE	
0	NO	AUMS			shift counts	
1	NO	AUMS				
2	YES	IAQ				
3	YES	GM		YES	first word of 2-word opcode; * LDD and LDS internal flags	
4	NO	AUMS			scratch register for EVAD *	
5	NO	AUMS		YES	one-word opcode or second * word of two-word opcode	
6	YES	SOP				
7	YES	DOP	YES		Td not 0	EVAD * destination address
		WS			Td = 0	
8	YES	SOP	YES		Ts not 0	EVAD * source address
		WS			Ts = 0	
9	YES	WS	YES		EVAD address of external * dest. register if *R+	
A	YES	WS	YES		EVAD address of external * source register if *R+	
B	NO	AUMS			BL and XOP	
C	YES	DOP			CRU base address	
D	YES	WS		YES	old WP	
E	YES	IOP		YES	old PC	
F	NO	AUMS		YES	old ST	

*EVAD should only be used if the WP=0000.

TABLE 10 — BUS STATUS CODE ASSOCIATED WITH WP VALUE
[WP bits 11 to 15 not equal to zero]

REGISTER ADDRESS BITS 11 TO 15	MAIN MEMORY ACCESS	BUS STATUS CODE
0	NO	AUMS
2	NO	AUMS
4	YES	IAQ
6	YES	GM
8	NO	AUMS
A	NO	AUMS
C	YES	SOP
E	YES	DOP
		WS
10	YES	SOP
		WS
12	YES	WS
14	YES	WS
16	NO	AUMS
18	YES	DOP
1A	YES	WS
1C	YES	IOP
1E	NO	AUMS

7.3.3.4 *Accessing Main Memory*

During Macrostore execution, data in the main memory is accessed using the indirect autoincrement and indexed addressing modes (*R, *R + and @TABLE(R)). MRAM workspace registers 2, 3, 6, 7, 8, 9, 10, 12, 13 and 14 are used as base registers during these accesses. This is only true when the Workspace Pointer resides on a 32-byte boundary, i.e., five LSB's = 0. When a routine residing in Macrostore accesses the main memory through one of these registers, the access is accompanied by a bus status code indicating a particular type of memory cycle, and $\overline{\text{MEM}}$ is held active-low. The bus status code corresponding to the use of each register is indicated in Table 9 for the case when WP = >0000. When the WP does not equal zero, the type of bus cycle and corresponding bus status code is determined by the least-significant addresses of the workspace register as shown in Table 10. For simplicity, it is recommended that the Workspace Pointer point to a 32-byte boundary to avoid confusion as to the type of bus cycle that will occur when the register is used as base register for memory transfers. Each main memory access should utilize a base register whose use is accompanied by the bus status code appropriate to the type of access being performed.

As shown in Table 9, WRs 7 and 8 are a special case in regard to the bus status code output during a main memory access. The default bus status code output by the processor is DOP (Table 2) when WR7 is used to access main memory; the default for WR8 is SOP. If an EVAD operation (to be described) is performed on an opcode whose Td field is 0 (workspace register direct addressing), the bus status code associated with WR7 is changed from its default of DOP to WS. Similarly, if an EVAD operation is performed on an opcode whose Ts field is 0, the bus status code for WR8 is changed from SOP to WS. If an EVAD subsequently is performed on an instruction with non-zero Td, the default of DOP is restored to WR7, and a non-zero Ts restores SOP to WR8. Everytime Macrostore is entered, the default status codes are restored.

Two examples illustrate the main memory access capability. The convention is to refer to the program in main memory that contains the MID opcode as the "user's" program. Assume that WR13 in MRAM contains the user's Workspace Pointer. To read the contents of WR4 in the user's workspace into WR1 in MRAM, the instruction MOV @8(R13), R1 is executed from Macrostore. A WS bus status code is output during this operation (MEM = 0, BST = 110). Second, assume that WR14 in MRAM contains the user's PC value. To read immediate data or a symbolic address (following a MID opcode in the user's program) into WR14 in MRAM, the instruction MOV *R14+,R1 is executed. This also causes the user's PC value in WR14 to be incremented by two, and an IOP bus status code is output (MEM = 0, BST = 010).

Using register 0, 1, 4, 5, 11 or 15 as base register for indirect autoincrement or indexed addressing results, in an access of Macrostore. During Macrostore accesses, the AUMS and AUMSL bus status codes are output to distinguish them from accesses of the main memory.

While executing in Macrostore, all symbolic addresses refer to locations within Macrostore. A I/O access using the base address in Macrostore register WR12 is accompanied by the I/O bus status code.

7.3.3.5 *Evaluate Address Instruction—EVAD*

An EVAD instruction during Macrostore execution permits convenient calculation of effective source and destination addresses for MID opcodes. EVAD assumes that the MID opcode contains a six-bit source operand field, and a six-bit destination operand field, i.e., the dual-operand format described in Section 10.5.1. The address calculations are based upon the original WP of the user, saved in WR13 in MRAM. Note that the EVAD instruction assumes that the WP is equal to zero as initialized upon entry into Macrostore. If the WP is modified to point to external MRAM, the WP must be restored to zero prior to EVAD execution. If the contents of a register in the user's workspace are fetched as part of the address calculation, a WS bus status code is output by the processor while the external access takes place. The saved PC (in WR14) is incremented appropriately if symbolic or indexed addressing is used. The contents of any workspace register in MRAM except WR0 can be evaluated using EVAD. When EVAD is executed, the calculated effective source address is placed in WR8 in MRAM, and the calculated destination address in WR7. If the source or destination field specifies autoincrement mode, the address of the user's register is placed in WR9 or WR10, respectively. Execution of EVAD alters the contents of WR4, which EVAD uses as a scratch register. A summary of the EVAD instruction, including its effect on status bits 0 and 2, is presented in Table 11.

Instruction	0 9	10 11	12 15
Format	0000 0001 00	Ts	S
For EVAD:	opcode	mode	register

The Ts and S fields above are used to determine the effective source address of the target word of the EVAD instruction. Once the target word is fetched as the source operand, the Ts, S, Td and D fields from that word are extracted and utilized as indicated below.

TABLE 11 – EVALUATE ADDRESS INSTRUCTION

STATUS BITS AFFECTED	DESCRIPTION
      0 → ST2 1 → ST0	DA → WR7 SA → WR8 If target Ts = (symbolic or indexed) WR14+2 → WR14 If target Td = 2 (symbolic or indexed) WR14+2 → WR14 If target Ts not 3 If target Td not 3
1 → ST2	If target Ts = 3 (autoincrement): address of external register → WR10
0 → ST0	If target Td = 3 (autoincrement): address of external register → WR9

The processor's WP register must be set to >0000 before executing the EVAD instruction. Otherwise, the results are unpredictable.

If only the source field of an MID opcode is to be evaluated, the Td field (bits 4 and 5) should be cleared to prevent unnecessary external accesses or unintentional modification of WR13 (generally the user's PC). For example, if the MID opcode resides in WR5 and bits 4 and 5 are not zero in MRAM, the instruction sequence

```
ANDI R5, >F3FF
EVAD R5
```

is executed while in Macrostore to calculate the effective source address. The destination field, which is all zeros, is interpreted as register direct addressing mode (and ST0 is set to one). In order to deal with the case where the source field specifies autoincrement mode (ST2 = 1), the instructions above are followed by

```
JEQ $ +4
INCT *R10
```

The second instruction increments the user's base register by two, assuming the source operand is one word in length. If the operand occupies a byte or double-word instead, the base register should be incremented by one or four, respectively.*

When developing an effective address based upon one of the user's workspace registers (in main memory), the EVAD instruction uses the contents of Macrostore register WR13. When developing an operand address based upon the user's program counter, the EVAD instruction uses the contents of Macrostore WR14. Note that WR14 is incremented by two for each symbolic or indexed addressing mode utilized.

7.3.3.6 *Jump on Interrupt Status*

The TB (test bit), SBO (set bit to one) and SBZ (set bit to zero) instructions are not available during Macrostore execution. In place of these operations and using the same opcodes are conditional jump instructions that detect pending interrupts. A "pending" interrupt is defined as an interrupt that has been requested by activating the processor's $\overline{\text{NMI}}$, or by asserting a request for an external interrupt that is not disabled by the interrupt mask in ST12-ST15. The instructions described in Table 12 allow interrupts to be tested at interruptible points in Macrostore routines. With this capability, instructions requiring long execution times can be emulated in a way that permits them to be interrupted and resumed after interrupt servicing.

The "jump if interrupt present" can be used to test for the occurrence of an interrupt.

The "jump not equal and no interrupt present" is useful in testing for interrupts while in loops. This single instruction may be used to exit a loop either on the condition that the loop count is zero or the interrupt is present.†

```
EXAMPLE:        LOOP    MOV  *R1+,*R2+
                        DEC  R3                  R3 HAS LOOP COUNT
                        SBZ  LOOP                DONE?
                        JNE  OUT                 JUMP TO OUT IF NO INTERRUPT
                         •
                         •
                         •
                OUT
```

These jump instructions have a displacement range of − 128 to + 127 words from the memory-word address following the jump instruction. The displacement is specified in the odd byte of each instruction. No status bits are affected by execution of a jump instruction.

The SBO and SBZ opcodes are executed in Macrostore as conditional jump instructions. SBO is equivalent to "jump if an interrupt is pending," and SBZ is equivalent to "jump if an interrupt is pending and ST2 is zero." The TB opcode is undefined in Macrostore. These instructions are summarized in Table 12.

TABLE 12—JUMP ON PENDING INTERRUPT

MNEMONIC	OPCODE	MEANING
SBO	1DXX	Jump if unmasked interrupt is present
SBZ	1EXX	Jump if equal bit is not set and unmasked interrupt is not present
TB	1FXX	Undefined

* The incrementing of workspace registers in the main memory is not performed by the EVAD Instruction but is the responsibility of the Macrostore software. Care may therefore be required to deal with the instance where the target word of an EVAD operation contains source and destination fields that specify indirect autoincrement using the same workspace register n (i.e., *Rn+, *Rn+). Otherwise, both the source and destination operands (pointed to by register n) will be read from the same address rather than from successive addresses.

† When using SBZ to check for exiting a loop, a JNE or JEQ instruction should follow (outside the loop) to determine the reason the loop was exited; SBO should not be used for this purpose when an interrupt is applied and then removed.

7.3.4 Subroutine Branch and Return

While executing in Macrostore, the BLWP instruction can be used to transfer program control to a subroutine located within Macrostore. For this purpose the opcode >0381 should be used. This version of the RTWP opcode should be distinguished from the RTWP variants >0380, >0382 and >0384, discussed in Section 7.2.6, all three of which cause an exit from Macrostore.

7.3.5 MID Opcodes in Interrupt Routines

One restriction exists regarding the use of MID opcodes within interrupt service routines. An MID opcode encountered in the interrupt routine for an NMI or level-1 interrupt, or for a Reset routine that does not cause complete system reinitialization, must not result in an exit from Macrostore by means of opcode >0382, the special form of RTWP that causes a level-2 trap. The reason is that the level-2 routine can be interrupted by an NMI, level-1 interrupt or Reset, possibly destroying the return linkage established previously. In general, this restriction can be interpreted to mean that a MID opcode in the service routine of an interrupt of higher priority than level 2 must either be recognized by an attached processor or defined by an emulation routine in Macrostore.

7.3.6 Testing for External Macrostore

The on-chip Macrostore software can use the following technique to allow the user to optionally expand the Macrostore functions by adding new routines residing in off-chip RAM or ROM. The TMS99110 uses this technique to check for populated off-chip Macrostore memory.

When the emulation software in the 99110's on-chip MROM determines that it cannot execute a particular MID opcode, it then checks to determine whether the system contains external Macrostore (off-chip RAM or ROM). If so, the Macrostore program branches to location >1002, the entry point of the emulation software in the external Macrostore. Otherwise, a level-2 interrupt is requested, as described in Section 7.2.4.1.

The check to determine whether the system contains external Macrostore works as follows. In a system having external Macrostore, the code >AAAA (alternative ones and zeros) must be stored at Macrostore address >1000, which is the first location in the off-chip region of Macrostore. The internal Macrostore emulation software upon deciding to test for external Macrostore, reads the contents of address >1000. If this location contains the code >AAAA, this confirms that the external Macrostore is present.

8. ATTACHED PROCESSOR (AP) INTERFACE

The TMS99000's basic instruction set can be extended by defining new instructions. The extended instruction set is supported either by emulation software contained in external Macrostore, or by external hardware utilizing the TMS99000's attached processor (AP) interface. The TMS99000's AP interface provides complete software transparency between these two methods. System support for extended instructions can be conveniently upgraded from Macrostore emulation routines to attached processors without affecting the user's software base.

An AP in a TMS99000 system attaches to the local bus of the microprocessor. While the processor is actively executing instructions, the AP passively monitors the bus to detect opcode fetches. The TMS99000 outputs an IAQ (instruction acquisition) bus status code to notify the AP each time an opcode fetch cycle occurs, and the AP latches the opcode from the bus to examine it. When the TMS99000 fetches an opcode which it does not recognize, but which the AP is prepared to execute, the TMS99000 transfers control of the local bus to the AP. After the AP completes execution of the instruction, it returns control to the processor.

The signals utilized by the AP interface of the TMS99000 are shown in Figure 29. The transfer of control from the TMS99000 to an AP and the eventual return of control to the TMS99000 takes place chiefly through the following three signals:

- $\overline{\text{APP}}$ (attached processor present) input
- MID (macro-instruction detected) bus status code
- HOLDA (hold acknowledge) bus status code

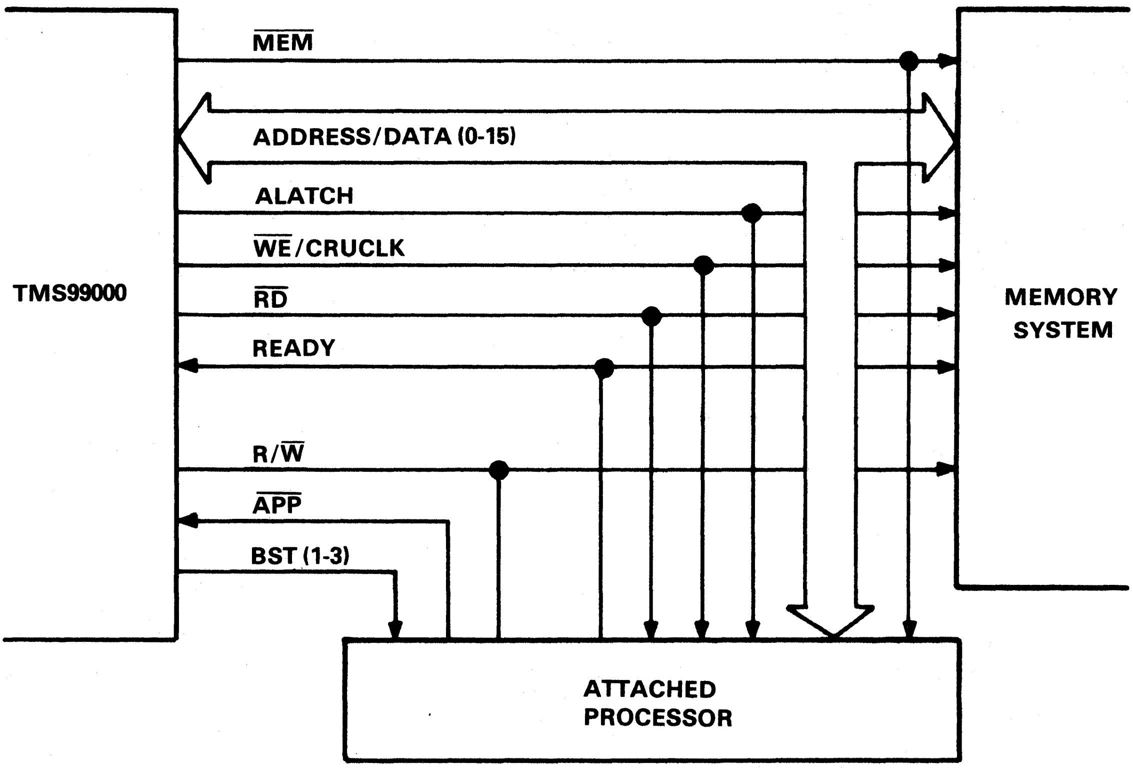
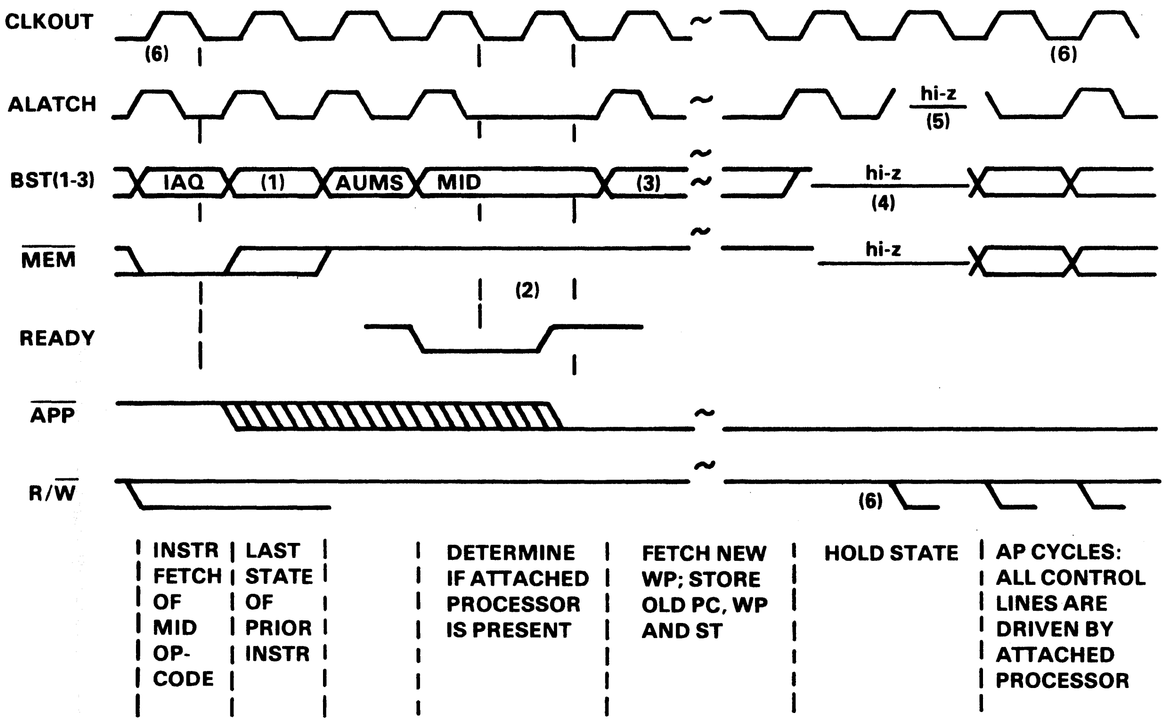


FIGURE 29—ATTACHED PROCESSOR INTERFACE

System memory, shared by the TMS99000 and the AP, is used to transfer context information from one to the other. The TMS99000's workspace registers, which reside in memory, are readily available to the AP while the AP remains in control of the local bus.

The timing for the AP interface is shown in Figures 30A and 30B. When the TMS99000 fetches an opcode it does not recognize, it outputs an MID bus status code to notify APs, should they be present, that it is prepared to relinquish system control. An opcode that causes this to occur will be referred to as an MID opcode. A list of MID opcodes is presented in Section 10.5.17. If bit 11 of the status register is set to 1, an XOP will also be treated as an MID opcode.


CLKOUT
(6)
(6)
ALATCH
hi-z
(5)
BST(1-3)
IAQ
(1)
AUMS
MID
(3)
hi-z
(4)
MEM
hi-z
(2)
READY
APP
R/W
(6)
INSTR FETCH OF MID OP-CODE
LAST STATE OF PRIOR INSTR
DETERMINE IF ATTACHED PROCESSOR IS PRESENT
FETCH NEW WP; STORE OLD PC, WP AND ST
HOLD STATE
AP CYCLES: ALL CONTROL LINES ARE DRIVEN BY ATTACHED PROCESSOR

NOTES:

(1) This bus status is determined by the prior instruction.

(2) Processor will remain in this state until READY goes high.

(3) BST = ST when the new status is output
= INTA when the new WP is fetched
= WP when the new WP is output
= WS while the old WP, PC, and ST are stored
(For simplicity, AUMS bus status codes are not shown.)

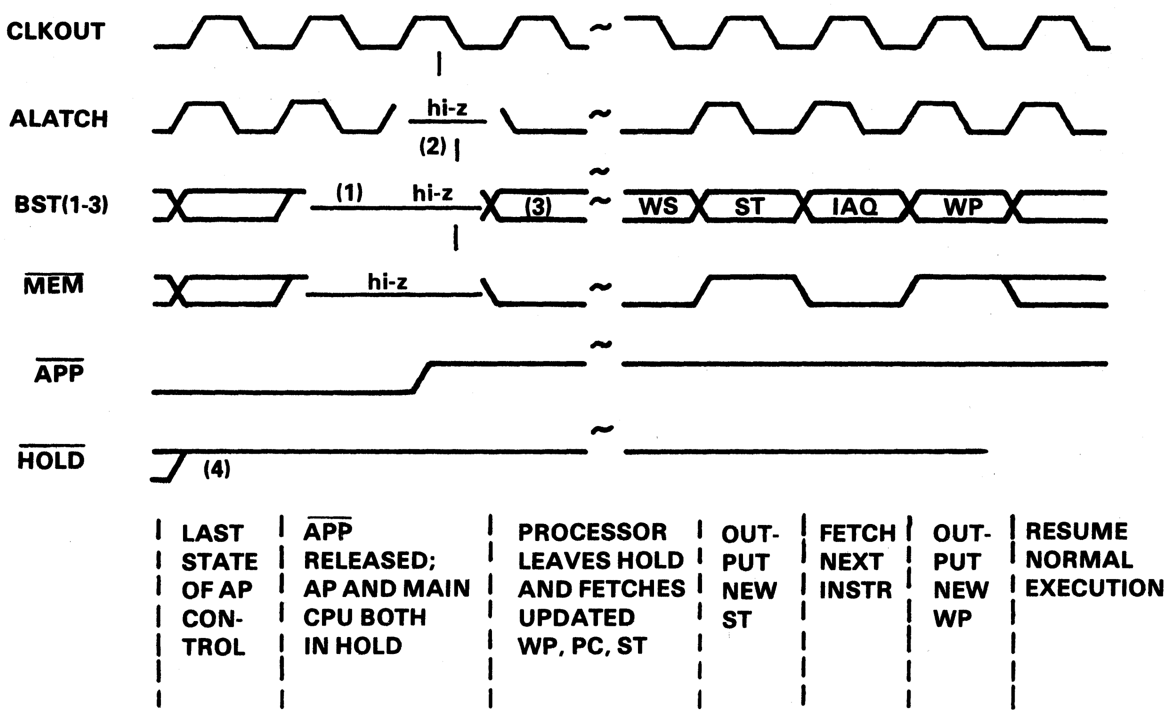
(4) The processors tristates all signals except ALATCH as follows:
- BST1-BST3 are first driven high to indicate hold acknowledge and then are tristated.
- MEM, RD and WE are first driven high and then tristated.
- The address-data bus is tristated "as is".

(5) The processor first drives its ALATCH output high and then tristates it.

(6) The CLKOUT remains the system clock throughout.

FIGURE 30—AP INTERFACE TIMING

(A) Transferring Control to Attached Processor



NOTES:

(1) The AP tristates all signals except ALATCH as follows:
- BST-1-BST-3 are first driven active high and then are tristated.
- MEM, RD and WE are first driven high and then are tristated.
- The address-data lines are tristated.

(2) The AP drives its ALATCH high and then tristates it.

(3) BST = WS during WP, PC, and ST fetches.

(4) An AP that fetches instructions for chained operations will assert HOLD and release APP during the instruction fetch to allow APP to be used for a breakpoint request.

FIGURE 30 – AP INTERFACE TIMING
(B) Regaining Control From Attached Processor

Assuming that an AP is (1) present and is (2) prepared to execute the MID opcode, it responds to the MID bus status code by pulling the APP line low to signify its readiness. Upon detecting the APP signal, the processor prepares to transfer control to the AP. This involves clearing status bit 8 and performing a context change. With the PSEL output signal high, the processor fetches the new WP value from the trap vector for the level 2 interrupt. (The PC value from the vector is not fetched.) The old WP, PC and ST values are saved in WRs 13, 14 and 15 of the new workspace. The saved PC points to the word following the MID opcode. After completing these actions, the processor begins a hold cycle, forces its outputs to the high-impedance state, and asserts HOLDA. This is the processor's signal that it is ready for the AP to assume control of the local bus.

Since the 99000 uses the same HOLDA bus status code to respond to both DMA devices and APs, each AP must monitor the HOLD line to distinguish a HOLDA in response to APP from a HOLDA in response to HOLD.

After taking control of the local bus, the AP begins executing the operation specified by the MID opcode. If a multiple-word instruction format is specified, the PC value saved in WR14 is used by the AP to access immediate data and operand address information. The contents of the original workspace are accessed through the WP value saved in WR13. The ST value in WR15 is altered to reflect the results of the operation performed.

The 99000 continually samples its APP and HOLD inputs during the hold cycle. When the AP completes its operation and releases APP, the processor responds by terminating the hold cycle. The processor loads PC, WP and ST registers with the values in WRs 13, 14 and 15, and resumes execution.

If an MID opcode is detected and $\overline{\text{APP}}$ remains high, indicating that no AP is prepared to execute the instruction, the processor performs a context switch that transfers control to the instruction emulation software contained in its Macrostore (Section 7).

The $\overline{\text{APP}}$ input performs a second function apart from its use in transferring control to an AP. An external device can use $\overline{\text{APP}}$ to force the processor to enter a hold cycle by asserting $\overline{\text{APP}}$ during the instruction acquisition (IAQ) cycle. The mechanism works as follows. The processor samples $\overline{\text{APP}}$ at the end of every opcode fetch, at the same point that it latches the opcode. The processor fetches the WP value from the level 2 trap vector and saves the old WP, PC, and ST values in WRs 13, 14, and 15 of the new workspace. The PC value saved in WR14 points to the memory word containing the opcode that was just fetched (and discarded). Following the context switch, the processor outputs the HOLDA bus status code, enters the hold state, and waits for $\overline{\text{APP}}$ to be released, as before.

The $\overline{\text{APP}}$ signal can be used by a maintenance panel to force the processor to enter hold. Using the mechanism described above, the maintenance panel can trigger $\overline{\text{APP}}$ on either a selected address or a selected opcode to cause a breakpoint. To avoid possible interference with APs, the maintenance panel should not assert $\overline{\text{APP}}$ during an MID bus status code if it was not active at IAQ. If the "panel option" is used with $\overline{\text{APP}}$, an attached processor should not assert $\overline{\text{APP}}$ until it has recognized a MID bus status code.

The processor acknowledges an unmasked interrupt upon completing execution of the instruction during which the interrupt becomes active. If the processor must respond to an interrupt before it can begin execution of a prefetched opcode,* the opcode is discarded prior to trapping to the interrupt service routine. Upon return from the interrupt, the opcode previously discarded is again fetched from memory. A special case of this procedure occurs when the discarded opcode is an MID opcode that an AP is preparing to execute. The AP must discard the opcode also. The AP knows to discard the opcode if the processor, following its fetch of the MID opcode, outputs the INTA bus status code. Alternatively to checking for the INTA bus status code, the AP can check for a subsequent IAQ bus status code indicating that the instruction has been discarded. This means that the processor has discarded the opcode in order to service the pending interrupt.

APs must monitor $\overline{\text{HOLD}}$ to detect DMA requests as discussed above. In a processor system containing one or more APs, the TMS99000 HOLDA signal is not distributed directly to DMA devices but is gated with the hold acknowledge signals from the APs to form a composite hold acknowledge signal that is passed on to the DMA devices. This composite hold acknowledge signal, which signifies transfer of control to the DMA device, is generated only after the processor and all APs have entered the hold state.

When an X (execute) instruction is executed, an IAQ bus status code is NOT output during the fetch of the target opcode located at the effective source address of the X instruction. Instead, an SOP or WS bus status code is output, depending on the addressing mode used. This means that APs cannot rely upon the IAQ bus status code to notify them when the processor fetches a MID opcode that is the target opcode of an X instruction.

The AP interface can be disabled by tying $\overline{\text{APP}}$ to ground. When operating in this mode, the processor automatically generates an ILLOP interrupt request upon encountering an MID opcode, bypassing the AP interface and Macrostore.

*The processor routinely prefetches the next opcode one state prior to completion of the current instruction (Section 10.6.2).

9. PIN DESCRIPTION

Table 13 defines the TMS99105A/TMS99110A pin assignments and describes the functions of each pin. Figure 31 illustrates the TMS99105A/TMS99110A pin assignment information.

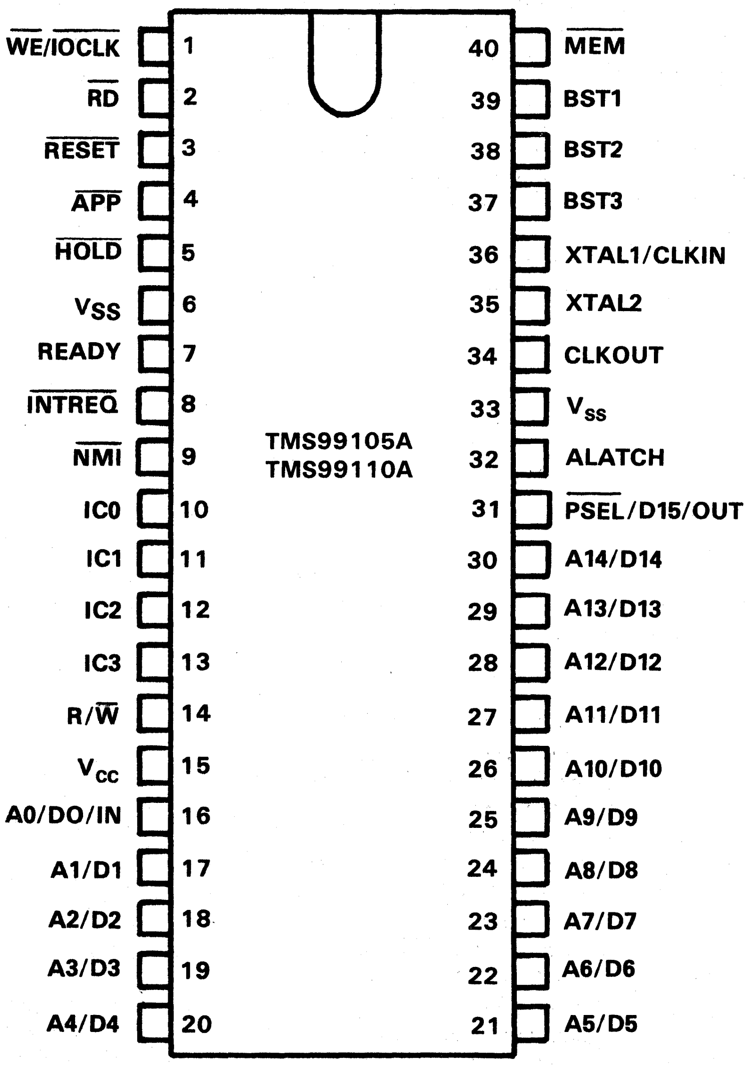


FIGURE 31 – PIN ASSIGNMENTS

TABLE 13—PIN DESCRIPTION

SIGNATURE	PIN	I/O	DESCRIPTION
			POWER SUPPLIES
V_{CC}	15		Supply voltage: + 5 V nominal.
V_{SS}	6, 33		Ground reference.
			CLOCKS
XTAL1/CLKIN	36	IN	Crystal input pin for internal oscillator; also input pin for external oscillator.
XTAL2	35	IN	Crystal input pin for internal oscillator.
CLKOUT	34	OUT	Clock output signal. The frequency of CLKOUT is ¼ the frequency of the crystal oscillator.
			ADDRESS/DATA BUS
A0/D0/IN (addr/data MSB)	16	I/O	While ALATCH = 1, these lines function as an address bus consisting of output signals A0-A14 and $\overline{\text{PSEL}}$. During memory, I/O and Macrostore accesses, an address is output on A0-A14. During memory cycles, status bit 8 is output in complemented form on $\overline{\text{PSEL}}$; $\overline{\text{PSEL}}$ is forced high during I/O accesses. During WP and ST bus cycles (Table 2), status information is output on the address bus. While ALATCH = 0, these lines function as a bidirectional data bus for memory, I/O and Macrostore accesses. During a bit-parallel byte or word read operation, ($\overline{\text{RD}}$ active low), data is input on D0-D15. During a bit-parallel write operation ($\overline{\text{WE}}/\overline{\text{IOCLK}}$ active low), data is output on D0-D15. For bit-serial I/O operations, read data is input on IN, and write data is output on OUT. These lines are forced to the high-impedance state during a hold cycle.
A1/D1	17	I/O	
A2/D2	18	I/O	
A3/D3	19	I/O	
A4/D4	20	I/O	
A5/D5	21	I/O	
A6/D6	22	I/O	
A7/D7	23	I/O	
A8/D8	24	I/O	
A9/D9	25	I/O	
A10/D10	26	I/O	
A11/D11	27	I/O	
A12/D12	28	I/O	
A13/D13	29	I/O	
A14/D14 (addr LSB)	30	I/O	
$\overline{\text{PSEL}}$/D15/OUT	31	I/O	
			LOCAL BUS CONTROL SIGNALS
ALATCH	32	OUT	Address latch. While ALATCH is high, the multiplexed address-data lines function as an address bus; while ALATCH is low, they function as a data bus. Each bus cycle (memory, I/O or internal) begins with a positive ALATCH pulse, the falling edge of which is used by external logic to latch the contents of the address bus. The $\overline{\text{MEM}}$ and BST1-BST3 outputs are stable while ALATCH is low. Prior to entering hold, the HOLDA bus status code is output and the ALATCH signal undergoes one final high-to-low transition before being driven to the high-impedance state. This permits an external device to latch the HOLDA code.
$\overline{\text{MEM}}$	40	OUT	Memory cycle. When low, $\overline{\text{MEM}}$ indicates that a memory cycle is in progress. When high, $\overline{\text{MEM}}$ indicates that a I/O or internal cycle is in progress. $\overline{\text{MEM}}$ is forced to the high-impedance state during a hold cycle; an internal resistive pull-up maintains a high level.
$\overline{\text{WE}}/\overline{\text{IOCLK}}$	1	OUT	Write enable and inverted I/O clock. When low, $\overline{\text{WE}}/\overline{\text{IOCLK}}$ indicates that write data is present on the data bus. $\overline{\text{WE}}/\overline{\text{IOCLK}}$ is active during memory writes ($\overline{\text{MEM}}$ = 0), serial I/O writes ($\overline{\text{MEM}}$ = 1, BST2 = 1, A0 = 0), parallel I/O writes ($\overline{\text{MEM}}$ = 1, BST2 = 1, A0 = 1), and writes to external Macrostore ($\overline{\text{MEM}}$ = 1, BST2 = 0). $\overline{\text{WE}}/\overline{\text{IOCLK}}$ is a tri-state output signal, and is forced to the high-impedance state during a hold state; an internal resistive pull-up maintains a high level.
$\overline{\text{RD}}$	2	OUT	Read Enable. When active low, $\overline{\text{RD}}$ indicates that a read (memory, parallel I/O, serial I/O or external Macrostore) is taking place on the bus, and that external devices may enable their tristate drivers to gate data onto the address-data lines. $\overline{\text{RD}}$ is a tristate signal and is forced to the high-impedance state during a hold state; an internal resistive pull-up maintains a high level.
R/$\overline{\text{W}}$	14	OUT	READ/WRITE. The R/$\overline{\text{W}}$ is valid at the beginning of each new cycle. This signal is high during read operation and low during write operations and internal ALU cycles. When R/$\overline{\text{W}}$ is low, it indicates that the 99000 will be driving the data bus. When R/$\overline{\text{W}}$ is high, it indicates that the 99000 will tristate the data bus (AD bus during the data time).

TABLE 13—PIN DESCRIPTION (CONTINUED)

SIGNATURE	PIN	I/O	DESCRIPTION
			LOCAL BUS CONTROL SIGNALS (CONCLUDED)
READY	7	IN	Ready. When high, READY indicates that the current bus cycle (memory, I/O or internal) is ready to be completed. As long as READY remains low to indicate a not ready condition, the bus cycle continues to be extended with wait states. Near the end of each wait state, READY is sampled to determine whether the bus cycle can complete or another wait state is to be generated. Note that this READY function differs from some READY functions in that bus cycles of non-memory cycles are affected by its operation.
			INTERRUPTS
$\overline{\text{INTREQ}}$	8	IN	Interrupt request. When active low, $\overline{\text{INTREQ}}$ indicates that an external interrupt is requested. If $\overline{\text{INTREQ}}$ is active, the processor latches the contents of the interrupt code inputs IC0-IC3 into its internal interrupt code register. The code is compared with the interrupt mask in status register bits 12-15. If the code is less than or equal to the mask value, the interrupt is granted; otherwise, the request is ignored. IC0-IC3 continue to be sampled as long as $\overline{\text{INTREQ}}$ remains low. If the request is initially disabled by the mask, $\overline{\text{INTREQ}}$ may be held low until the mask changes to a value that enables the request.
IC0 (MSB)	10	IN	Interrupt code. IC0 is the MSB of the 4-bit interrupt code. IC0-IC3 are sampled when $\overline{\text{INTREQ}}$ is active low. The highest-priority interrupt level is signified by IC0-IC3 = LLLL; the lowest level is HHHH.
IC1	11	IN	
IC2	12	IN	
IC3 (LSB)	13	IN	
$\overline{\text{NMI}}$	9	IN	Non-maskable interrupt. When active low, $\overline{\text{NMI}}$ causes the processor to perform a non-maskable interrupt using the trap vector located at memory address FFFC. The $\overline{\text{NMI}}$ sequence begins following the execution of the instruction in progress at the time the $\overline{\text{NMI}}$ request is initiated. The $\overline{\text{NMI}}$ will also terminate an idle state. If $\overline{\text{NMI}}$ is active during the time $\overline{\text{RESET}}$ is released, the $\overline{\text{NMI}}$ sequence will occur following completion of the reset sequence, but prior to execution of the first instruction in the reset service routine. $\overline{\text{NMI}}$ must be active for at least one CLKOUT cycle to be recognized and will only be recognized once for each high-to-low transition.
$\overline{\text{RESET}}$	3	IN	$\overline{\text{RESET}}$. When active low, $\overline{\text{RESET}}$ causes the processor to set all status bits to zero and inhibits $\overline{\text{WE}}$/$\overline{\text{IOCLK}}$, $\overline{\text{RD}}$ and $\overline{\text{MEM}}$ internally. When $\overline{\text{RESET}}$ is released, the processor initiates a level 0 interrupt sequence using the trap vector at memory address 0000, clears the entire status register, and begins executing the reset service routine. $\overline{\text{RESET}}$ also will terminate an idle state. $\overline{\text{RESET}}$ must be held active for at least three CLKOUT periods to guarantee that a Reset will take place. $\overline{\text{RESET}}$ is a Schmitt-trigger input.
			DMA REQUEST
$\overline{\text{HOLD}}$	5	IN	$\overline{\text{HOLD}}$. An external controller generates a hold request by pulling the processor's $\overline{\text{HOLD}}$ input low. This indicates the controller's wish to obtain control of the local bus to perform one or more DMA transfers. The processor responds to the hold request by outputting a HOLDA bus status code (Table 12) and then forcing $\overline{\text{MEM}}$, $\overline{\text{WE/IOCLK}}$, $\overline{\text{RD}}$, BST1-BST3, R/$\overline{\text{W}}$, ALATCH and the address data lines to the high-impedance state. When $\overline{\text{HOLD}}$ is released, the processor terminates the hold cycle and resumes processing.
			BUS STATUS
BST1 (MSB)	39	OUT	Bus status lines. These lines are used with the $\overline{\text{MEM}}$ output to provide external circuitry with information concerning the nature of the bus cycle currently in progress. The bus status codes are presented in **Table 2.** MPILCK is indicated by BST1-BST3 = 000. BST1-BST3 are forced to the high-impedance state during a hold cycle.
BST2	38	OUT	
BST3 (LSB)	37	OUT	

TABLE 13—PIN DESCRIPTION (CONCLUDED)

SIGNATURE	PIN	I/O	DESCRIPTION
ATTACHED PROCESSOR			
$\overline{\text{APP}}$	4	IN	Attached processor present. When the TMS99000 fetches an MID opcode (Section 2.4), it outputs an MID bus status code and samples the $\overline{\text{APP}}$ input. If $\overline{\text{APP}}$ has been pulled low by an external device, the CPU performs a context switch and relinquishes control of the local bus. The CPU fetches the new WP from the level 2 trap vector, and the old WP, PC, and ST are saved in the new workspace. The CPU signals its release of the local bus by outputting a HOLDA bus status code and then enters hold. After the attached processor has completed its operation, it releases $\overline{\text{APP}}$; the CPU responds by terminating, restoring its context, and resuming processing. If no external device asserts $\overline{\text{APP}}$, the CPU attempts to emulate the MID opcode in Macrostore and traps to the level 2 interrupt service routine if the opcode is undefined in Macrostate.

10. INSTRUCTION SET

10.1 DEFINITION

Each TMS99000 instruction performs one of the following:

- Arithmetic or logical operation on data, or comparison or manipulation of data,
- Loading or storing of internal registers (program counter, workspace pointer, or status register),
- Data transfer between memory and external devices via the I/O, or
- Control functions.

10.2 ADDRESSING MODES

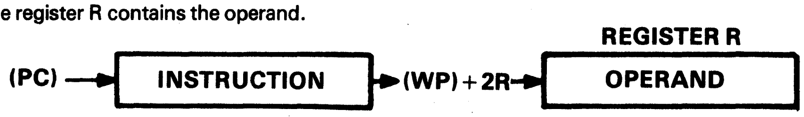
The TMS99000 instruction set provides a variety of modes for addressing random memory data, e.g., program parameters and flags, or formatted memory data (character strings, data lists, etc.). These addressing modes are:

- Workspace register addressing
- Workspace register indirect addressing
- Workspace register indirect autoincrement addressing
- Symbolic (direct) addressing
- Indexed addressing
- Immediate addressing
- Program counter relative addressing
- I/O relative addressing

The derivation of the effective address for each addressing mode is described graphically below. The applicability of each addressing mode to particular instructions is described in Section 10.5, along with the operation performed by each instruction. The symbols following the names of the addressing modes, R, *R, *R+, @LABEL and @TABLE(R), are the general forms used by processor assemblers to specify the addressing mode for workspace register R.

10.2.1 Workspace Register Addressing, R

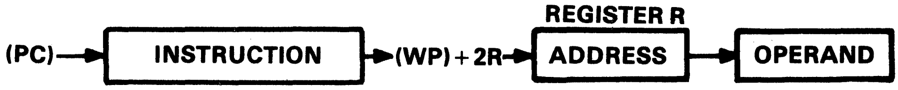
Workspace register R contains the operand.



The workspace register addressing mode is specified by setting the two-bit T-field (Ts or Td) of the instruction word to 0.

10.2.2 Workspace Register Indirect Addressing, *R

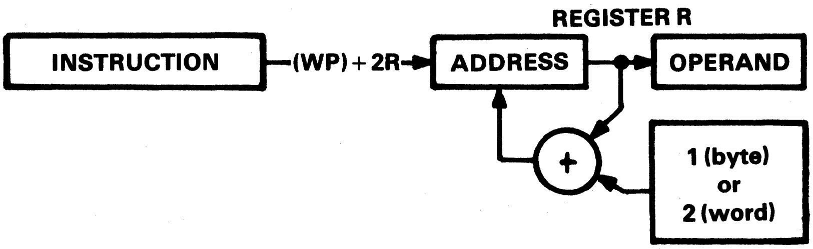
Workspace register R contains the address of the operand.



The workspace register indirect addressing mode is specified by setting the two bits in the T-field (Ts or Td of the instruction word to 01.

10.2.3 Workspace Register Indirect Autoincrement Addressing, *R+

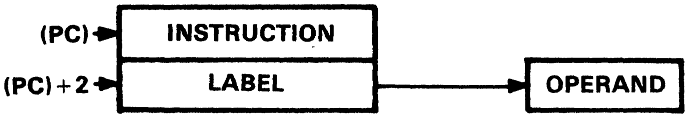
Workspace register R contains the address of the operand. After acquiring the address of the operand, the contents of the workspace register are incremented.



The workspace register indirect autoincrement addressing mode is specified by setting the two-bit T-field (Ts or Td) of the instruction word to 3.

10.2.4 Symbolic (Direct) Addressing, @LABEL

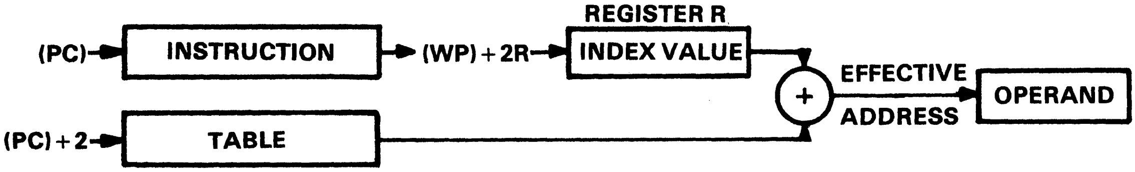
The word following the instruction contains the address of the operand.



The symbolic addressing mode is specified by setting the two-bit T-field (Ts or Td) of the instruction word to 2 and setting the corresponding S or D field equal to 0.

10.2.5 Indexed Addressing, @TABLE(R)

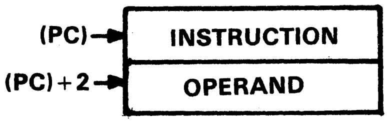
The word following the instruction contains the base address. Workspace register R contains the index value. The sum of the base address and the index value results in the effective address of the operand.



The indexed addressing mode is specified by setting the two-bit T-field (Ts or Td) of the instruction word to 2 and setting the corresponding S or D field to a value other than 0. The value in the S or D field is the number of the workspace register which contains the index value.

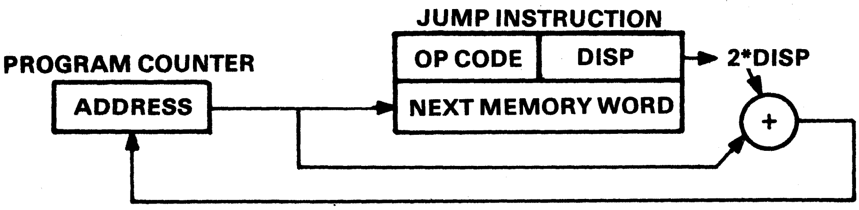
10.2.6 Immediate Addressing

The word following the instruction contains the operand.



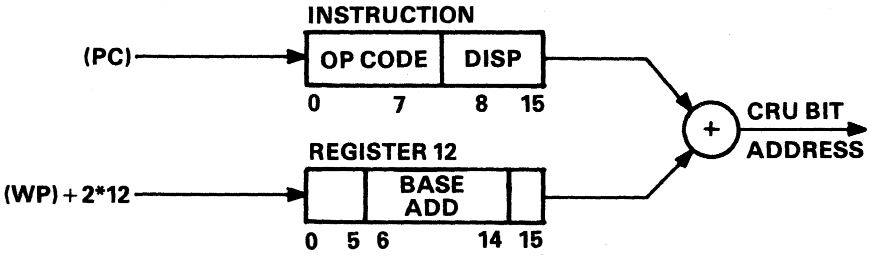
10.2.7 Program Counter Relative Addressing

The 8-bit signed displacement in the right byte (bits 8 through 15) of the instruction is multiplied by 2 and added to the updated contents of the program counter. The result is placed in the PC.



10.2.8 I/O Relative Addressing

The 8-bit signed displacement in the right byte of the instruction is added to the I/O base address (bits 0 through 14 of workspace register 12). The result is the address of the selected bit in I/O space.



10.3 TERMS AND DEFINITIONS

The terms used in describing the instructions of the processor are defined in Table 14.

TABLE 14—SYMBOL CONVENTIONS

SYMBOL	DEFINITION
B	Byte indicator (1 = byte; 0 = word)
C	Bit count
D	Destination address register
DA	Destination address
IOP	Immediate operand
LSB(n)	Least-significant (rightmost) bit of n
MSB(n)	Most-significant (leftmost) bit of n
N	Don't care
PC	Program counter
result	Result of operation performed by instruction
S	Source address register
SA	Source address
ST	Status register
STn	Bit n of status register
Td	Destination address modifier
Ts	Source address modifier
W	Workspace register
WRn	Workspace register n
(n)	Contents of n
((n))	Indirect contents of n
a → b	A is transferred to b
|n|	Absolute value of n
+	Arithmetic addition
−	Arithmetic subtraction
AND	Logical AND
OR	Logical OR
⊕	Logical exclusive OR
$\overline{n}$	Logical complement of n
•	Arithmetic multiplication
I/O base address	The address which is stored in WR12
effective I/O base address	The address which is formed by adding the displacement to the base address in WR12 for single bit I/O, or the incremented value of WR12 for multibit I/O.
I/O bit address	The effective address of a bit located in the lower half of the I/O space.

10.4 STATUS REGISTER MANIPULATION

Various TMS99000 machine instructions affect the status register. Figure 5 shows the status register bit assignments. Table 15 lists the instructions and their effect on the status register.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS*

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST0	LOGICALLY GREATER THAN	C, CB	If MSB(SA) = 1 and MSB(DA) = 0, or if MSB(SA) = MSB(DA) and MSB((DA) – (SA)) = 1
		CI	If MSB(W) = 1 and MSB of IOP = 0, or if MSB(W) = MSB of IOP and MSB(IOP – (W)) = 1
		ABS, LDCR	If (SA) is not zero
		RTWP	If bit 0 of WR15 is 1
		LST	If bit 0 of selected WR is 1
		A, AB, AI AM, ANDI, DEC, DECT, LI, MOV, MOVB, NEG, ORI, S, SB, DIVS, MPYS, INC, INCT, INV, SLA, SLAM, SM, SOC, SOCB, SRA, SRAM, SRC, SRL, STCR, SZC, SZCB, XOR	If result is not 0 (see Note 2)
		Reset	ST0 is cleared unconditionally
		All other instructions and interrupts	ST0 is not affected (see Note 1)

*See Table 13 for definition of terminology used.

NOTES: 1. The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.
2. If on a DIVS instruction an overflow occurs, ST4 is set and ST0, ST1, and ST2 are undefined.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS (CONTINUED)

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST1	ARITHMETIC GREATER THAN	C, CB	If MSB(SA) = 0 and MSB(DA) = 1, or if MSB(SA) = MSB(DA) and MSB((DA) – (SA)) = 1
		CI	If MSB(W) = 0 and MSB of IOP = 1, or if MSB(W) = MSB of IOP and MSB(IOP – (W)) = 1
		ABS, LDCR	If MSB(SA) = 0 and (SA) is not 0
		RTWP	If bit 1 of WR15 is 1
		LST	If bit 1 of selected WR is 1
		A, AB, AI, AM, ANDI, DEC, DECT, LI, MOV, MOVB, NEG, ORI, S, SB, DIVS, MPYS, INC, INCT, INV, SLA, SLAM, SM, SOC, SOCB, SRA, SRAM, SRC, SRL, STCR, SZC, SZCB, XOR	If MSB of result = 0, and result is not 0 (see Note 2)
		Reset	ST1 is cleared unconditionally
		All other instructions and interrupts	ST1 is not affected (see Note 1)

NOTES: 1. The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.
2. If on a DIVS instruction an overflow occurs, ST4 is set and STO, ST1, and ST2 are undefined.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS (CONTINUED)

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST2	EQUAL/TB INDICATOR	C, CB	If (SA) = (DA)
		CI	If (W) = IOP
		COC	If ((SA) and not (DA)) = 0
		CZC	If ((SA) and (DA)) = 0
		TB	If CRUIN = 1 for addressed CRU bit
		TSMB, TCMB, TMB	If addressed memory bit = 1
		ABS, LDCR	If (SA) = 0
		RTWP	If bit 2 of WR15 is 1
		LST	If bit 2 of selected WR is 1
		A, AB, AI, AM, ANDI, DEC, DECT, LI, MOV, MOVB, NEG, ORI, S, SB, DIVS, MPYS, INC, INCT, INV, SLA, SLAM, SM, SOC, SOCB, SRA, SRAM, SRC, SRL, STCR, SZC, SZCB, XOR	If result = 0 (see Note 2)
		Reset	ST2 is cleared unconditionally
		All other instructions and interrupts	ST2 is not affected (see Note 1)
ST3	CARRY OUT	A, AB, ABS, AI, AM, DEC, DECT, INC, INCT NEG, S, SM, SB	If carry out = 1
		SLA, SRA, SRL, SRC, SRAM, SLAM	If last bit shifted out = 1
		RTWP	If bit 3 of WR15 is 1
		LST	If bit 3 of selected WR is 1
		Reset	ST3 is cleared unconditionally
		All other instructions and interrupts	ST3 is not affected (see Note 1)

NOTES: 1. The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.
2. If on a DIVS instruction an overflow occurs, ST4 is set and ST0, ST1, and ST2 are undefined.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS (CONTINUED)

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST4	ARITHMETIC FAULT	A, AB, AM	If MSB(SA) = MSB(DA) and MSB of result ≠ MSB(DA)
		AI	If MSB(W) ≠ MSB of IOP and MSB of result ≠ MSB(W)
		S, SB, SM	If MSB(SA) = MSB(DA) and MSB of result = MSB(DA)
		DEC, DECT	If MSB(SA) = 1 and MSB of result = 0
		INC, INCT	If MSB(SA) = 0 and MSB of result = 1
		SLA, SLAM	If MSB changes during shift
		DIV	If MSB(SA) = 0 and MSB(DA) = 1, or if MSB(SA) = MSB(DA) and MSB((DA) – (SA)) = 0
		DIVS	If the quotient cannot be expressed as signed 16-bit quantity (>8000 is a valid negative number)
		ABS, NEG	If (SA) = >8000
		RTWP	If bit 4 of WR15 is 1
		LST	If bit 4 of selected WR is 1
		Reset	ST4 is cleared unconditionally
		All other instructions and interrupts	ST4 is not affected*
ST5	PARITY (ODD NO. OF "1" BITS)	CB, MOVB	If (SA) has odd number of ones
		LDCR	If C = 1 to 8 and (SA) has odd number of ones (if C = 9 to 15 or C = 0, then ST5 is not affected)
		AB, SB, SOCB, SZCB,	If result has odd number of ones
		STCR	If C = 1 to 8 and the result has an odd number of ones (if C = 0 or C = 9 to 15, then ST5 not affected)
		RTWP	If bit 5 of WR15 is 1
		LST	If bit 5 of selected WR is 1
		Reset	ST5 is cleared unconditionally
		All other instructions and interrupts	ST5 is not affected (See Note 1)

NOTES: 1. The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.
2. If on a DIVS instruction an overflow occurs, ST4 is set and ST0, ST1, and ST2 are undefined.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS (CONTINUED)

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST6	XOP IN PROGRESS	XOP	If XOP instruction is executed (ST6 set after the context switch)
		RTWP*	If executed when ST7 = 1 (non-privileged mode), then ST6 is cleared
		LST*	If executed when ST7 = 1 (non-privileged mode), then ST6 is cleared.
		Reset	ST6 is cleared unconditionally
		All other instructions and interrupts	ST6 is not affected (see Note 1)
ST7	PRIVILEGED MODE	RTWP*	If bit 7 of WR15 is 1
		LST*	If bit 7 of selected WR is 1
		XOP, any interrupt	ST7 is cleared unconditionally
		All other instructions	ST7 is not affected (see Note 1)
ST8	MAP SELECT	RTWP*	If bit 8 of WR15 is 1
		LST*	If bit 8 of selected WR is 1
		XOP, any interrupt	ST8 is cleared unconditionally prior to read of trap vector. Previous value is saved in WR15.
		LDCR, STCR, SBO, SBZ, TB	ST8 temporarily driven to 0 while CRU address is on the address bus
		All other instructions	ST8 is not affected (see Note 1)
ST9	UNDEFINED	RTWP*	If bit 9 of WR15 is 1
		LST*	If bit 9 of selected WR is 1
		XOP, any interrupt	ST9 is cleared unconditionally
		All other interrupts	Do not affect status bit (see Note 1)
ST10	ARITHMETIC FAULT INTERRUPT ENABLE	RTWP*	If bit 10 of WR15 is 1
		LST*	If bit 10 of selected WR is 1
		XOP, any interrupt	ST10 is cleared unconditionally
		All other instructions	ST10 is not affected (see Note 1)

*Status bits 7, 8, 9, 11, 12, 13 and 14 are not affected by LST or RTWP if ST7 = 1 before these instructions are executed.

Note 1. The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.

2. If on a DIVS instruction an overflow occurs, ST4 is set and ST0, ST1, and ST2 are undefined.

TABLE 15 – STATUS REGISTER BIT DEFINITIONS (CONCLUDED)

BIT	NAME	INSTRUCTION	CONDITIONS TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST11	XOP EMULATION MODE	RTWP*	If bit 11 of WR15 is 1
		LST*	If bit 11 of selected WR is 1
		XOP, any interrupt	ST11 is cleared unconditionally
		All other instructions	ST8 is not affected
ST12 to ST15	INTERRUPT MASK	LIMI†	Set mask = bits 12-15 of IOP
		RTWP*	Set mask = bits 12-15 of WR15
		LST*	Set mask = bits 12-15 of WR
		RSET† RESET, NMI	Mask is unconditionally cleared (set to all zeros)
		All other interrupts	If mask = 0, no change; otherwise, set mask to interrupt level minus one.
		All other instructions	Mask is not affected (see Note 1)

*Status bits 7, 8, 9, 11, 12, 13, and 14 are not affected by LST or RTWP if ST7 = 1 before these instructions are executed.

†ST12 to ST15 are not affected by LIMI and RSET if ST7 = 1.

NOTE 1: The X instruction itself does not set any status bits, but the target instruction may set status bits accordingly.

10.5 INSTRUCTIONS

A list of the instructions described in each of the following subsections is presented below for convenient reference.

Instruction Mnemonic	Section
A, AB, C, CB, S, SB, SOC, SOCB, SZC, SZCB, MOV, MOVB	10.5.1
COC, CZC, XOR, MPY, DIV	10.5.2
MPYS, DIVS	10.5.3
XOP	10.5.4
B, BL, BLWP, CLR, SETO, INV, NEG, ABS, SWPB, INC, INCT, DEC, DECT, X	10.5.5
BIND	10.5.6
LDCR, STCR	10.5.7
SBO, SBZ, TB	10.5.8
JEQ, JGT, JH, JHE, JL, JLE, JLT, JMP, JNC, JNE, JNO, JOC, JOP	10.5.9
SLA, SRA, SRC, SRL	10.5.10
AI, ANDI, CI, LI, ORI, BLSK	10.5.11
LWPI, LIMI	10.5.12
STST, LST, STWP, LWP	10.5.13
RTWP	10.5.14
IDLE, RSET, CKOF, CKON, LREX	10.5.15
TMB, TCMB, TSMB	10.5.16
AM, SM, SLAM, SRAM	10.5.17
MID opcodes	10.5.18
LDD, LDS	Appendix B
LR, STR, NR, CER, CIR, CRE, CRI, AR, DR, SR, MR	Appendix B

10.5.1 Dual-Operand Instructions with Multiple Addressing for Source and Destination Operand

General	0 2	3	4 5	6 7 8 9	10 11	12 13 14 15
Format:	OPCODE	B	Td	D	Ts	S

If B = 1, the operands are bytes and the effective operand addresses are byte addresses. If B = 0, the operands are words and the LSB of each effective operand address is ignored.

The addressing mode for each operand is determined by the two bits of the T-field corresponding to that operand.

Ts or Td	S or D	ADDRESSING MODE	NOTES
0	0,1, . . . ,15	Workspace register	1
1	0,1, . . . ,15	Workspace register indirect	
2	0	Symbolic	4
2	1,2, . . . ,15	Indexed	2,4
3	0,1, . . . ,15	Workspace register indirect autoincrement	3

NOTES:
1. When a workspace register is the operand of a byte instruction (bit 3 = 1), the left byte (bits 0 through 7) is the operand and the right byte (bits 8 through 15) is not altered.
2. Workspace register 0 may not be used for indexing.
3. The workspace register is incremented by 1 for byte instructions (bit 3 = 1) and is incremented by 2 for word instructions (bit 3 = 0).
4. When Ts = Td = 2, two words are required in addition to the instruction word. The first word is the source operand and the second word is the destination operand base address.

MNEMONIC	OPCODE 0123	MEANING	COMPARED TO 0	RESULT BITS AFFECTED	STATUS DESCRIPTION
A	1010	Add	Yes	0-4	(SA) + (DA) →(DA)
AB	1011	Add bytes	Yes	0-5	(SA) + (DA) →(DA)
C	1000	Compare	No	0-2	Compare (SA) to (DA) and set appropriate status bits
CB	1001	Compare bytes	No	0-2,5	Compare (SA) to (DA) and set appropriate status bits
S	0110	Subtract	Yes	0-4	(DA) – (SA) → (DA)
SB	0111	Subtract bytes	Yes	0-5	(DA) – (SA) →(DA)
SOC	1110	Set ones corresponding	Yes	0-2	(DA) OR (SA) →(DA)
SOCB	1111	Set ones corresponding bytes	Yes	0-2,5	(DA) OR (SA) →(DA)
SZC	0100	Set zeros corresponding	Yes	0-2	(DA) AND (SA) →(DA)
SZCB	0101	Set zeros corresponding	Yes	0-2,5	(DA) AND (SA) →(DA)
MOV	1100	Move	Yes	0-2	(SA) →(DA)
MOVB	1101	Move bytes	Yes	0-2,5	(SA) →(DA)

10.5.2 Dual-Operand Instructions with Multiple Addressing Modes for the Source Operand and Workspace Register Addressing for the Destination

General	0 1 2 3 4 5	6 7 8 9	10 11	12 13 14 15
Format:	OPCODE	D	Ts	S

The addressing mode for the source operand is determined by the Ts field.

Ts	S	ADDRESSING MODE	NOTES
0	0,1, . . . ,15	Workspace register	
1	0,1, . . . ,15	Workspace register indirect	
2	0	Symbolic	
2	1,2, . . . ,15	Indexed	1
3	1,2, . . . ,15	Workspace register indirect autoincrement	2

NOTES:
1. Workspace register 0 may not be used for indexing.
2. The workspace register is incremented by 2.

MNEMONIC	OPCODE 012345	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
COC	001000	Compare ones corresponding	No	2	Test (D) to determine if 1s are in each bit position where 1s are in (SA). If so, set ST2.
CZC	001001	Compare zeros corresponding	No	2	Test (D) to determine if 0s are in each bit position where 1s are in (SA). If so, set ST2.
XOR	001010	Exclusive OR	Yes	0-2	(D) + (SA) → (D)
MPY	001110	Multiply	No	– –	Multiply unsigned (D) by unsigned (SA) and place unsigned 32-bit product in D (most significant) and D + 1 (least significant). If WR15 is D, the next word in memory after WR15 is used for the least significant half of the product.
DIV	001111	Divide	No	4	If unsigned (SA) is less than or equal to unsigned (D), perform no operation and set ST4. Otherwise, divide unsigned (D) and (D + 1) by unsigned (SA). Quotient → (D), remainder →(D + 1) If D = 15, the next word in memory after WR15 will be used for the remainder.

10.5.3 Signed Multiply and Divide Instructions

General Format:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	OPCODE										Ts		S			

The addressing mode for the source operand is determined by the Ts field.

Ts	S	ADDRESSING MODE	NOTES
00	0,1, . . . ,15	Workspace register	1
01	0,1, . . . ,15	Workspace register indirect	1
10	0	Symbolic	1
10	1,2, . . . ,15	Indexed	1,2
11	1,2, . . . ,15	Workspace register indirect autoincrement	1,3

NOTES: 1. Workspace registers 0 and 1 contain operands used in the signed multiply and divide operations.

2. Workspace register 0 may not be used for indexing.

3. The workspace register is incremented by 2.

MNEMONIC	OPCODE 0123456789	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
MPYS	0000000111	Signed Multiply	Yes	0-2	Multiply signed 2's complement integer in WRO by signed 2's complement integer in (SA) and place signed 32-bit product in WR0 (most significant) and WR1 (least significant).
DIVS	0000000110	Signed Divide	Yes	0-2,4	If the quotient cannot be expressed as a signed 16-bit quantity (hex 8000 is a valid negative number), set ST4. Otherwise, divide the signed, 2's complement integer in WRO and WR1 by the signed 2's complement integer at SA and place the signed quotient in WR0 and the signed remainder in WR1. The sign of the quotient is determined by algebraic rules. The sign of the remainder is the same as the sign of the dividend, and |REMAINDER| < |DIV|.

10.5.4 Extended Operation (XOP) Instruction

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	0	0	1	0	1	1	D				Ts		S			

The Ts and S fields provide multiple-mode addressing capability for the source operand.

Depending on the value of status bit 11 (ST11), the XOP instruction transfers control to a user routine located either at the main memory address in the specified XOP trap vector, or at Macrostore.

If ST11 = 0, the D field specifies the trap vector in memory that contains the addresses of the entry point and workspace of the user routine to be executed. The address of the trap vector is calculated as

>0040 + >4 × >D

Following the fetch of the new WP and PC values, the effective source address (SA) is calculated and placed in WR11 of the new workspace. The old WP, PC and ST are stored in WRs 13, 14 and 15, respectively. Status bit 6 is set to 1, and STs 7 through 11 are cleared after the old status has been saved.

When ST11 = 1, the XOP causes a trap to Macrostore if the 99000 is not in the baseline mode. The contents of the WP are forced to 0, and the PC is updated with the value contained at Macrostore address >0812. The new WP and PC point to locations within the Macrostore, where address space is logically distinct from the main memory address space. The old WP, PC and ST are stored in registers 13, 14 and 15, respectively, of the Macrostore workspace. Status bits 7 through 11 are cleared after the old status has been saved.

The execution of the XOP instruction is summarized below. If ST11 is 0, the addresses are memory addresses; if ST11 is 1 and the 99000 is not in baseline mode, the addresses are in Macrostore.

If ST11 is 0:	If ST11 is 1 Macrostore is entered and:
(0040+4×D) → WP	0 → WP
(0042+4×D) → PC	(0812) → PC
SA → new WR11	(old WP) → (new WR13)
(old WP) → (new WR13)	(old PC) → (new WR14)
(old PC) → (new WR14)	(old ST) → (new WR15)
(old ST) → (new WR15)	
1 → ST6 0 → ST9	
0 → ST7 0 → ST10	
0 → ST8 0 → ST11	

The TMS99000 does not test interrupt requests (i.e., does not look at $\overline{\text{INTREQ}}$) upon completion of the XOP instruction.

10.5.5 Single Operand Instructions

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	OPCODE										Ts		S			

The Ts and S fields provide multiple-mode addressing capability for the source operand.

MNEMONIC	0123456789	MEANING	RESULT TO 0?	BITS AFFECTED	DESCRIPTION
B	0000010001	Branch	No	– –	SA →(PC)
BL	0000011010	Branch and link	No	– –	(PC) →(WR11), SA →(PC)
BLWP	0000010000	Branch and load workspace pointer	No	– –	(SA) →(WP), (SA+2) →(PC), (old WP) →(new WR13), (old PC) →(new WR14), (old ST) → (new WR15). The $\overline{\text{INTREQ}}$ input is not tested upon completion of the BLWP instruction.
CLR	0000010011	Clear operand	No	– –	0 →(SA)
SETO	0000011100	Set to ones	No	– –	FFFF →(SA)
INV	0000010101	Invert	Yes	0-2	(SA) →(SA)
NEG	0000010100	Negate	Yes	0-4	– (SA) →(SA)
ABS	0000011101	Absolute value*	No	0-4	|(SA)| →(SA)
SWPB	0000011011	Swap bytes	No	– –	Bits 0-7 of (SA) →bits 8-15 of (SA); bits 8-15 of (SA) →bits 0-7 of (SA).
INC	0000010110	Increment	Yes	0-4	(SA) + 1 →(SA)
INCT	0000010111	Increment by two	Yes	0-4	(SA) + 2 →(SA)
DEC	0000011000	Decrement	Yes	0-4	(SA) – 1 →(SA)
DECT	0000011001	Decrement by two	Yes	0-4	(SA) – 2 →(SA)
X†	0000010010	Execute	No	– –	Execute instruction located at SA.

*Operand is compared to zero for status bit.

†If additional memory words for the execute instruction are required to define the operands of the instruction located at SA, these words will be accessed from PC and the PC will be updated accordingly. The IAQ (instruction acquisition) bus status code will not be ouput at the time the process reads the instruction at SA; instead, an SOP (source operand) or WS bus status code will be output. Status bits are affected in the usual manner for the operation performed.

10.5.6 BIND Instruction

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	OPCODE										Ts		S			

The Ts and S fields provide multiple-mode addressing capability for the source operand.

The BIND instruction serves as the inverse of a BLSK instruction if the register indirect autoincrement addressing mode is used. Indexed addressing used with BIND implements a powerful CASE or multi-way branch instruction where the immediate operand points to a table of branch addresses and the register contents selects which way to branch.

MNEMONIC	0123456789	MEANING	RESULT TO 0?	BITS AFFECTED	DESCRIPTION
BIND	0000000101	Branch indirect	No	– –	(SA)→(PC)

10.5.7 Multiple-Bit I/O Instructions

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	OPCODE						CNT				Ts		S			

The I/O base address is contained in bits 0 through 14 of WR12. If bit 0 (the MBS) of the base address is 0, a serial I/O transfer will occur; otherwise (MSB = 1), a parallel I/O transfer will occur.

In the case of a serial I/O transfer, the CNT field specifies the number of bits to be transferred (from 1 to 16). If CNT = 0, 16 bits are transferred. The base address in WR12 defines the starting I/O bit address. The bits are transferred in bit-serial fashion, and the I/O base address is incremented by 2 with each bit transfer; the contents of WR12 are not affected. The effective source address in memory, specified by the Ts and S fields, is interpreted as a byte address if 8 or fewer bits are transferred (CNT = 1 through 8), or as a word address if 9 or more bits are transferred (CNT = 0, 9 through 15). If the source is addressed in the workspace indirect autoincrement mode (Ts = 3), the specified workspace register is incremented by 1 if CNT is in the range 1 to 8, and is incremented by 2 otherwise. If the source is addressed in the register mode (Ts = 0), bits 8 through 15 of the specified workspace register are unchanged if the transfer is of 8 bits or less.

In the case of a parallel I/O transfer, the CNT field determines whether a byte or word is to be transferred, and also whether the contents of WR12 are to be incremented by 2 following the transfer. A word transfer occurs if CNT is (binary) 1010 or 1011; a byte transfer occurs if CNT is 0010 or 0011. WR12 is post-incremented by 2 if CNT is 0011 or 1010. All values of CNT besides 0010, 0011, 1010 and 1011 are reserved for future expansion of the parallel I/O capability and should not be used. The following table summarizes the use of the CNT field for a parallel I/O operation.

TRANSFER	CNT* (BINARY)	DESCRIPTION
byte transfer	0010	WR12 not altered
	0011	WR12 post-incremented by 1
word transfer	1010	WR12 not altered
	1011	WR12 post-incremented by 2

*These restrictions on the value of CNT apply only in the case of parallel I/O operations.

When in user mode (ST7 = 1), an attempt to execute an LDCR instruction having a I/O address in the range 1C00 to 7FFE or 9C00 to FFFE is flagged as a privileged opcode violation. This condition generates a level 2 interrupt and inhibits writes to the I/O in the privileged space for the duration of the instruction. When in privileged mode (ST7 = 0), the I/O address of an LDCR instruction is unrestricted. When in user mode (ST7 = 1), an attempt to execute an STCR with an I/O address 1C00 to 7FFE or 9C00 to FFFE causes a privileged violation to occur after execution of the instruction.

MNEMONIC	OPCODE 012345	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
LDCR	001100	Load communication register	Yes	0-2,5*	Beginning with LSB of (SA), transfer the specified number of bits from (SA) to the I/O.
STCR	001101	Store communication register	Yes	0-2,5*	Beginning with LSB of (SA), transfer the specified number of bits from the I/O to (SA). Load unfilled bit positions with 0.

*ST5 is affected only if CNT is in the range 1 to 8.

10.5.8 Single-Bit I/O Instructions

General Format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OPCODE								SIGNED DISPLACEMENT							

The signed displacement is added to the contents of WR12 (bits 0-14) to form the address of the I/O bit to be selected, as described in Section 8.2.8.

When in user mode (ST7 = 1), if the effective I/O address of an SBO or SBZ instruction is in the range >1C00 to >7FFE or >9C00 to > FFFE, a privileged violation occurs (Section 4.4.3) and the I/O write is inhibited. When in privileged mode (ST7 = 0), no restrictions are placed on the range of the effective I/O address.

The user is cautioned that while the SBO and SBZ instructions can be used to access the parallel I/O address space (>8000 to >FFFF), and they will set or clear data bit D15 as expected, the other 15 bits (D0 to D14) written to the parallel I/O location will be undefined. When the TB instruction is executed with an address in parallel I/O space, the bit value input on data line D0 is read.

When in Macrostore, the SBO, SBZ and TB instructions are not available. The SBO and SBZ opcodes perform different functions when in Macrostore (see Section 7.3.3.6).

MNEMONIC	OPCODE 0123 4567	MEANING	STATUS BITS AFFECTED	DESCRIPTION
SBO	0001 1101	Set bit to one	– –	Set the selected output bit to 1.
SBZ	0001 1110	Set bit to zero	– –	Set the selected output bit to 0.
TB	0001 1111	Test bit	2	If the selected I/O input bit is 1, set ST2; if the selected I/O input bit is 0, clear ST2.

10.5.9 Jump Instructions

General Format:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td colspan="8">OPCODE</td><td colspan="8">SIGNED DISPLACEMENT</td></tr>
</table>

Jump instructions cause the PC to be loaded with the PC-relative jump address if the selected status bits are set as specified; otherwise, no jump occurs and the next instruction is fetched from the word following the jump instruction. The jump address is computed by adding twice the signed displacement to the current value of the PC (which points to the word following the jump instruction). The 8-bit displacement permits the computed jump address to be specified anywhere in the range – 128 to + 127 words from the address of the word that follows the jump instruction. Status register bits are not affected by jump instructions.

MNEMONIC	OPCODE 01234567	MEANING	STATUS CONDITION TO LOAD PC
JEQ	00010011	Jump equal	ST2 = 1
JGT	00010101	Jump greater than	ST1 = 1
JH	00011011	Jump high	ST0 = 1 and ST2 = 0
JHE	00010100	Jump high or equal	ST0 = 1 or ST2 = 1
JL	00011010	Jump low	ST0 = 0 and ST2 = 0
JLE	00010010	Jump low or equal	ST0 = 0 or ST2 = 1
JLT	00010001	Jump less than	ST1 = 0 and ST2 = 0
JMP	00010000	Jump unconditional	Unconditional
JNC	00010111	Jump no carry	ST3 = 0
JNE	00010110	Jump not equal	ST2 = 0
JNO	00011001	Jump no overflow	ST4 = 0
JOC	00011000	Jump on carry	ST3 = 1
JOP	00011100	Jump odd parity	ST5 = 1

10.5.10 Shift Instructions

General Format:

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td colspan="8">OPCODE</td><td colspan="4">SC</td><td colspan="4">W</td></tr>
</table>

Field SC contains the shift count. W is the number of the workspace register whose contents are to be shifted. If SC = 0, however, bits 12 through 15 of WR0 are used as the shift count. If SC = 0 and bits 12 through 15 of WR0 are 0, the effective shift count is 16.

MNEMONIC	OPCODE 01234567	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
SLA	00001010	Shift left arithmetic	Yes	0-4	Shift (W) left. Fill vacated bit positions with 0.
SRA	00001000	Shift right arithmetic	Yes	0-3	Shift (W) right. Fill vacated bit positions with original MSB of (W).
SRC	00001011	Shift right circular			Shift (W) right. Shift previous LSB into MSB.
SRL	00001001	Shift right logical	Yes	0-3	Shift (W) right. Fill vacated bit positions with zeros.

10.5.11 Immediate Register Instructions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General	OPCODE												W			
Format:	IOP															

MNEMONIC	OPCODE 0123 45678 91011	MEANING	RESULT TO 0?	BITS AFFECTED	DESCRIPTION
AI	0000 00100 010	Add immediate	Yes	0-4	(W) + IOP → (W)
ANDI	0000 00100 100	AND immediate	Yes	0-2	(W) AND IOP → (W)
CI	0000 00101 000	Compare immediate	Yes	0-2	Compare (W) to IOP and set appropriate status bits.
LI	0000 00100 000	Load immediate	Yes	0-2	IOP → (W)
ORI	0000 00100 110	OR immediate	Yes	0-2	(W) OR IOP →
BLSK	0000 00001 011	Branch immediate and push link to stack	No	– –	(W) – 2 → (W), (PC) + 4 → ((W) IOP → (PC)

10.5.12 Internal Register Load Immediate Instructions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General	OPCODE											0	0	0	0	0
Format:	IOP															

When in user mode (ST7 = 1), execution of the LIMI instruction is flagged as a privileged opcode violation (Section 4.4.3).

MNEMONIC	OPCODE 0123 4567 8910	MEANING	DESCRIPTION
LWPI	0000 0010 111	Load workspace pointer immediate	IOP → (W) not status bits affected.
LIMI	0000 0011 000	Load interrupt mask immediate	IOP → ST bits 12 thru 15, ST12 thru ST15.

10.5.13 Internal Register Load and Store Instructions

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	OPCODE												W			

MNEMONIC	OPCODE 0123 4567 891011	MEANING	STATUS BITS AFFECTED	DESCRIPTION
STST	0000 0010 1100	Store status Register	– –	(ST) → (W)
LST	0000 0000 1000	Load status Register	0-15	(W) → (ST)
STWP	0000 0010 1010	Store workspace pointer	– –	(WP) → (W)
LWP	0000 0000 1001	Load workspace pointer	– –	(W) → (WP)

While in privileged mode (ST7 = 0), the LST instruction modifies all 16 bits of the status register. While in user mode (ST7 = 1), only bits 0 through 5 and bit 10 of the workspace register specified in the W field are placed in the status register; ST6 is cleared and the other status register bits are unaffected.

10.5.14 Return Workspace Pointer (RTWP) Instruction

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0

The RTWP instruction causes the following transfers to occur:

(WR15) → (ST)
(WR14) → (PC)
(WR13) → (WP)

When in privileged mode (ST7 = 0), the RTWP instruction causes the entire contents of WR15 to be loaded into the status register. In user mode (ST7 = 1), only bits 0 through 5 and 10 of WR15 are loaded into the status register; ST6 is cleared and the other status register bits remain unaffected.

When in Macrostore, several variations of the RTWP instruction opcode are available. These are opcodes >0381, >0382, and >0384. These opcodes are summarized below. More detail in the operation of these special opcodes is given in Section 7.3.2

RTWP Opcode	Function
>0380	RTWP when in main memory or exit from Macrostore with interrupts sampled
>0381	RTWP when in Macrostore memory (does not cause exit from Macrostore)
>0382	Exit from Macrostore with level 2 trap
>0384	Exit from Macrostore and suppress interrupt sample.

10.5.15 External Instructions

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	OPCODE											0	0	0	0	0

External instructions cause a bit value of 0 to be written to a I/O address 1ECX, where the hexadecimal digit represented as "X" depends upon the particular external instruction being executed. During execution of the RSET, CKOF, CKON and LREX instructions, the WE/IOCLK output is pulsed low once. With the completion of the single I/O write cycle, execution of the external instruction is finished, and the processor proceeds to the next instruction. While in privileged mode (ST7 = 0), execution of RSET causes the interrupt mask (ST12-ST15) to be cleared. None of the other external instructions affect the status register.

When the IDLE instruction is executed, the processor enters the idle state, where it remains until a Reset, NMI, APP, or unmasked external interrupt occurs. While in the idle state, the processor pulses the WE/IOCLK output repeatedly, with each I/O write cycle accompanied by a I/O bus status code (Table 2). The PC value saved during the context switch to the Reset, NMI or interrupt service routine points to the instruction following the IDLE.

When in user mode (ST7 = 1), execution of an external instruction is flagged as a privileged opcode violation (Section 4.4.3).

MNEMONIC	OPCODE 0123 4567 8910	MEANING	STATUS BITS AFFECTED	DESCRIPTION	I/O ADDRESS IN HEX
IDLE	0000 0011 010	Idle	– –	Suspend processor instruction execution until an interrupt, NMI or Reset occurs.	1EC4
RSET	0000 0011 011	Reset	12-15	Clear interrupt mask (ST12-ST15)	1EC6
CKOF	0000 0011 110	User-defined	– –	– –	1ECC
CKON	0000 0011 101	User-defined	– –	– –	1ECA
LREX	0000 0011 111	User-defined	– –	– –	1ECE

10.5.16 Bit-Manipulation Instructions

General Format:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	OPCODE															
	0	0	0	0	0	0	BIT DISP				Ts		S			

The Ts and S fields provide multiple-mode addressing capability for the source operand. The indirect autoincrement addressing mode (Ts = 3), however, is undefined for the TMB, TCMB and TSMB instructions. If the two bits of the Ts field are 3, an MID trap occurs.

Bit-manipulation instructions copy the specified memory bit into status bit 2, and set or clear the specified memory bit. In order to provide an indivisible test-and-set operation, the MPILCK (multiprocessor interlock) bus status code is active during the critical portions of the TSMB and TCMB instructions, except in the case Ts = 0 (register addressing mode).

MNEMONIC	OPCODE IN HEX	MEANING	STATUS BITS AFFECTED	DESCRIPTION*
TMB	0C09	Test memory bit	2	(SA+BD) → ST2
TCMB	0C0A	Test and clear memory bit	2	(SA+BD) → ST2, 0 → (SA+BD)
TSMB	0C0B	Test and set memory bit	2	(SA+BD) → ST2, 1 → (SA+BD)

*BD is used above to refer to the contents of the bit-displacement field.

If the leading 6 bits in the predefined field of the second word of the instruction are not as specified, an MID trap occurs.

10.5.17 Double-Precision Arithmetic Instructions

General Format:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ADD/SUB	OPCODE															
	0	1	0	0	Td		D				Ts		S			
SHIFT	OPCODE															
	0	1	0	0	0	0	SC				Ts		S			

MNEMONIC	OPCODE (HEX	MEANING	STATUS BITS AFFECTED	DESCRIPTION
AM	002A	Add double	0-4	(SA,SA+2)+(DA,DA+2) → (DA, DA+2)
SM	0029	Subtract double	0-4	(DA,DA+2) – (SA,SA+2) → (DA,DA+2)
SLAM	001D	Shift left arithmetic double	0-4	Shift (SA,SA+2) left. Fill vacated bit positions with 0. If SC=0, count is in bits 4 through 7 of WR0.
SRAM	001C	Shift right arithmetic double	0-3	Shift (SA,SA+2) right. Fill vacated bit positions with MSB. If SC=0, count is in bits 4 through 7 of WR0.

If the two bits in the Ts or Td field are 3 (workspace register indirect autoincrement addressing mode) the contents of the corresponding workspace register are incremented by 4.

If SC = 0 in the shift instructions, the shift count is taken from bits 4 through 7 of WR0, which are interpreted as an unsigned 4-bit integer. If bits 4 through 7 of WR0 are 0, then the effective shift count is 0. Bits shifted out are shifted into ST3. If the shift count is 0, ST3 is set to 0.

During a SRAM the sign bit fills the vacated positions. During a SLAM, zeros fill the vacated positions.

If the bits in the predefined field of the second word of the instruction are not as specified, an MID trap occurs.

10.5.18 MID Opcodes

The single-word instruction opcodes that cause an MID trap (see Section 4.4.2) are:

0000-001B
001E-0028
002B-007F
00A0-00AF
00C0-013F
0210-021F
0230-023F
0250-025F
0270-027F
0290-029F
02B0-02BF
02D0-02DF
02E1-02FF
0301-033F
0341-035F
0361-037F
0381-039F
03A1-03BF
03C1-03DF
03E1-03FF
0780-07FF
0C00-0C08
0C0C-0FFF

10.6 INSTRUCTION EXECUTION

10.6.1 Microinstruction Cycle

The TMS99000 microprocessor is a microcoded machine. Each instruction in the 99000 instruction set is executed internally as a sequence of microinstructions, the length of the sequence varying according to the particular instruction. Each microinstruction cycle is minimally one machine state in duration but can be extended with wait states by activating the READY input. The term "wait-state" is used to describe the condition where the processor is "frozen" in its present state and consequently cannot advance to the next state. In the 99000, all types of bus cycle – memory, I/O or internal – can be extended with wait-states. The ALATCH output toggles exactly once at the beginning of each microinstruction cycle.

10.6.2 Opcode Prefetching

The TMS99000 increases its effective processing speed by prefetching opcodes where possible. By allowing successive bus cycles to be overlapped, as shown in Figure 27, the time required to fetch the opcode from memory and decode it becomes transparent when no wait states are required. In processing a typical instruction, e.g., register-to-register add, the TMS99000 performs the following sequence of steps:

1. Fetch instruction
2. Decode instruction
3. Fetch source operand, if needed
4. Fetch destination operand, if needed
5. Process the operands
6. Store the results, if required

The prefetch mechanism of the 99000 makes use of the fact that the processor's memory interface can operate in parallel with operations involving the processor's internal buses and registers. For example, during step 5 above, the memory bus is not needed by the current instruction, which is busy processing the operands internally. Hence, this time can be used to prefetch the opcode for the next instruction. This overlapping is seen in Figure 32, where "MI" indicates an operation performed by the memory interface, and "OP" denotes an internal operation. Deterministic: a prefetched opcode is discarded only in the event that an interrupt occurs. Steps 1 and 2 above should really be considered part of the preceding instruction. In other words, each instruction is responsible for prefetching the opcode for the next instruction. This reduces the effective overhead of the typical instruction sequence given above to the four steps, 3-6. Without overlap, the overhead would be six rather than four steps.

The instruction prefetch scheme employed by the 99000 can cause self-modifying software to execute incorrectly. Incorrect execution results when one instruction attempts to generate the opcode of the very next instruction to be executed. The TMS99000 fetches the opcode of the next instruction before storing the result of the current instruction.

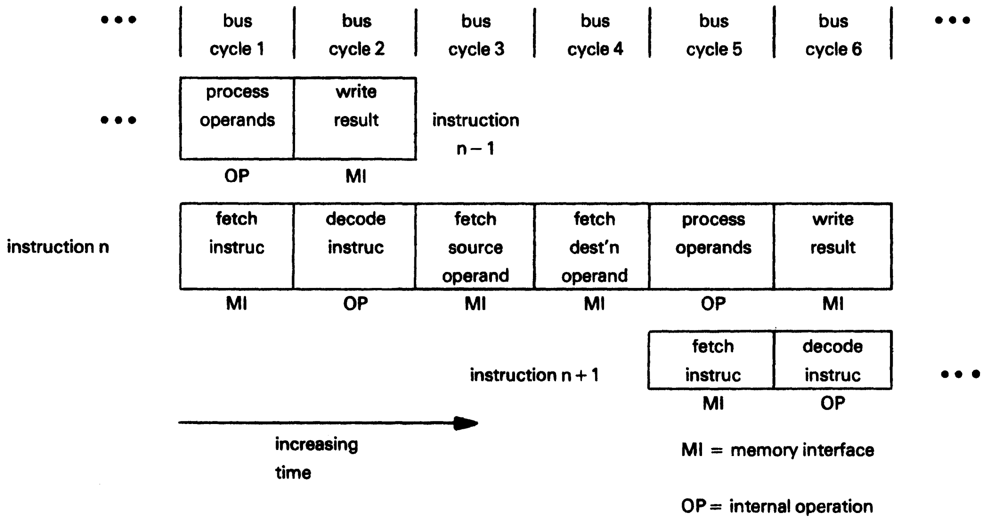


FIGURE 32 – OVERLAPPED INSTRUCTION EXECUTION

10.6.3 TMS99000 Instruction Execution Times

Instruction execution times for the TMS99000 are a function of the:

- Machine state time ts (four times the external input clock period),
- Particular addressing mode used in the event that the instruction provides multiple-mode addressing capability, and
- Number of wait states required per memory access.

Table 16* lists the number of machine states and memory accesses required to execute each 99000 instruction. For instructions providing multiple addressing modes for one or both operands, the table lists the number of states and memory accesses with all operands addressed in the workspace register mode. To determine the additional number of states and memory accesses required for the other addressing modes, add the appropriate values from the table. The total execution time for an instruction, assuming all memory requires the same number of wait states, is calculated as:

$$T = ts(C + WM)$$

where:

T = total instruction execution time
ts = machine state time (four times the external input clock period)
C = number of states for instruction execution plus address modification
W = number of required wait states per memory access for instruction execution plus address modification
M = number of memory accesses

For example, consider a MOV instruction executed in a system for which ts = 0.167 μsec. Assume that no wait states are required to access memory, and that both operands are accessed in workspace register mode:

$$T = ts(C + WM) = 0.167(3 + 0 \times 3)\ \mu sec = 0.50\ \mu sec$$

If two wait states per memory access are required, the execution time becomes

$$T = 0.167(3 + 2 \times 3)\ \mu sec = 1.50\ \mu sec$$

If the source operand was addressed in the symbolic mode and two wait states are required, then

$$T = tc(C + WM),$$
$$C = 3 + 1 = 4,$$
$$M = 3 + 1 = 4,$$
$$T = 0.167(4 + 2 \times 4)\ \mu sec = 2.0\ \mu sec$$

*Instruction prefetching is accounted for in Table 16. The table gives exact cycle counts required for instruction execution.

TABLE 16—INSTRUCTION EXECUTION TIMES

INSTRUCTIONS	MACHINE STATES C	MEMORY ACCESS M	ADDRESS MODIFICATION	
			SOURCE	DEST
A	4	4	A*	A
AB	4	4	A	A
ABS	5	3	A	—
AI	4	4	—	—
AM	12	8	A	A
ANDI	4	4	—	—
B	3	1	A	—
BIND	4	2	A	—
BL	5	2	A	—
BLSK	7	5	—	—
BLWP	10	6	A	—
C	4	3	A	A
CB	4	3	A	A
CI	4	3	—	—
CKON	9	1	—	—
CKOF	9	1	—	—
CLR	3	2	A	—
COC	4	3	A	—
CZC	4	3	A	—
DEC	3	3	A	—
DECT	3	3	A	—
DIV (ST4 is set)	6 or 10	4	A	—
DIV (ST4 is reset)†	30	6	A	—
DIVS (ST4 is set)	10,13 or 33	4	A	—
DIVS (ST4 is reset)†	34	6	A	—
IDLE	9+2×N	1	—	—
INC	3	3	A	—
INCT	3	3	A	—
INV	3	3	A	—
JUMP (PC is changed)	3	1	—	—
(PC is not changed)	3	1	—	—
LDCR (CNT=0, serial)	40	3	A	—
(CNT≠0, serial)	8+2×CNT	3	A	—
(MSB R12=1, autoincrement R12)	8	4	A	—
(MSB R12=1, R12 not autoincremented)	8	3	A	—
LDD and LDS‡				
LI	3	3	—	—
LIMI	5	2	—	—
LMF‡				
LREX	9	1	—	—
LST	7	2	—	—

* Replace the letter "A: with appropriate value from Table A. The C and M values from Table A for the addressing mode used must be added to the C and M values from this table.

† Execution time is dependent upon the partial quotient after each clock cycle during execution.

‡ Execution time is added to the execution time of the source address.

TABLE 16 – INSTRUCTION EXECUTION TIMES (CONCLUDED)

INSTRUCTIONS	MACHINE STATES	MEMORY ACCESS	ADDRESS MODIFICATION	
	C	M	SOURCE	DEST
LWP	3	2	—	—
LWPI	3	2	—	—
MOV	3	3	A*	A
MOVB	4	4	A	A
MPY	23	5	A	—
MPYS	25	5	A	—
NEG	3	3	A	—
ORI	4	4	—	—
RSET	9	1	—	—
RTWP¶	9/7	4	—	—
S	4	4	A	A
SB	4	4	A	A
SBO	7	2	—	—
SBZ	7	2	—	—
SETO	3	2	—	—
SHIFT (SC≠0)	5+SC	3	—	—
(SC=0 and bits 12-15 of WR=0)	23	4	—	—
(SC=0 and bits 12-15 of WR≠0)	7+SC	4	—	—
SM	12	7	A	A
SOC	4	4	A	A
SOCB	4	4	A	A
SHIFT MULTIPLE (SC=0)	11+SC	5	A	—
(SC≠0)	13+SC	6	A	—
STCR (CNT=0, serial)	43	3	A	—
(CNT=1 to 7)	20+CNT	4	A	—
(CNT=8)	27	4	A	—
(CNT=9 to 15)	20+CNT	3	A	—
(MSB R12=1, autoincrement R12)	10	5	A	—
(MSB R12=1, R12 not autoincremented)	10	4	A	—
STST	3	2	—	—
STWP	3	2	—	—
SWPB	3	3	A	—
SZC	4	4	A	A
SZCB	4	4	A	A
TB	7	2	—	—
TEST MEM BIT	26	3	A	—
X§	2	1	A	—
XOP (ST11=0)	14	7	A	—
XOR	4	4	A	—
Reset function	13	6	—	—
Interrupt context switch	13	6	—	—
MID opcode (Macrostore)	14§	0	—	—
(attached processor)	21§	8	—	—

* Replace the letter "A" with appropriate value from Table A. The C and M values from Table A for the addressing mode used must be added to the C and M values from this table.

§Execution time does not include the time required by software or an attached processor to emulate the instruction.

¶RTWP, when staying in Macrostore, takes seven machine states. When not in Macrostore or exiting Macrostore, RTWP takes nine machine states.

TABLE A

ADDRESSING MODE	MACHINE STATES C	MEMORY ACCESS M
WR (Ts or Td = 0)	0	0
WR indirect (Ts or Td = 1)	1	1
WR indirect autoincrement (Ts or Td = 3)	3	2
Symbolic (Ts or Td = 2, S or D = 0)	1	1
Indexed (Ts or Td = 2, S or D = 0)	3	2

10.6.4 Bus Status Code Sequences

This section presents the sequence of bus status codes output by the microprocessor during each (1) instruction execution, (2) interrupt trap, (3) MID trap and (4) transfer of control between the TMS99000 and an attached processor.

The TMS99000 microprocessors are microcoded machines. Each instruction in the instruction set is executed internally as a sequence of microinstructions, the length of the sequence varying according to the particular instruction. Each microinstruction cycle is minimally one machine state in duration but can be extended with wait states by activating the READY input. The sequence of machine states generated during the execution of a particular instruction depends upon the opcode and the addressing modes used.

A typical instruction contains an opcode and addressing modes for up to two operands (source and destination). The execution of an instruction can similarly be divided into two parts: (1) the derivation of the operands from the specified addressing modes, and (2) the execution of the operation specified by the opcode. Since the same addressing modes are common to many instructions, the portion of the execution sequence corresponding to each addressing mode can be listed separately from the basic execution sequences for the various instructions. The listing of these sequences in separate tables is done in this section for the sake of brevity.

Using this information, the user can reconstruct the entire sequence for a particular instruction by inserting the sequences corresponding to the addressing modes into the basic sequence given for the instruction. The basic execution sequences for the various TMS99000 instructions are presented in Table 14. In this table, the sequences corresponding to the source and destination addressing modes are represented by the symbols <SRC> and <DST>, respectively. These symbols must be replaced by the appropriate sequences from Tables 15 and 16 to reconstruct the entire execution sequence for the instruction with its particular addressing modes.

An example will help to illustrate this procedure. Consider the following add instruction:

A *R1+,R2

The addressing mode used to locate the source operand is register indirect autoincrement with R1. The addressing mode used to locate the destination operand is register direct with R2. Table 14 presents a complete list of the machine states generated during the execution of this instruction. Each state is identified by the bus status code output during that state. Refer to Table 2 of Section 3 for a list of all bus status codes and their mnemonics. The fetching of the A (add) opcode is not shown in Table 14; instead, the next to the last state shown is the prefetch of the opcode for the instruction that follows the add. This convention will be followed throughout Table 17. The prefetch of the opcode for the next instruction is considered to be part of the execution sequence of the current instruction.

Using the data presented in Tables 18, 19 and 20, the information presented in the example of Table 17 is constructed as follows. The basic execution sequence for the A (add) instruction is presented at the beginning of Table 18. Here the execution sequences for the source and destination addressing modes are represented by the symbols <SRC> and <DST>. These symbols are replaced by the appropriate addressing mode sequences from Table 18 to generate the sequence seen in Table 17. The symbols Ns and Nd in Table 17 represent the number of machine cycles required to derive the source and destination operands, respectively, and are replaced by the appropriate numbers from Table 18.

The execution sequences for all other TMS99000 instructions and operations shown in Table 17 are generated in similar fashion.

TABLE 17—EXAMPLE INSTRUCTION SEQUENCE FOR AN A *R1+,R2

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
Ns=4	0110	WS	R	Fetch source operand from WR1
	1001	AUMS	—	Internal operation
	0110	WS	W	Increment WR1
	0001	SOP	R	Read source operand
Nd=1	0110	WS	R	Read dest'n operand from WR2
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Write sum to WR2

TABLE 18—SOURCE ADDRESSING MODE SEQUENCES

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT	
Workspace Register Source Addressing, R					
1	0110	WS	R	Get source operand from WR	Ns = 1
Workspace Register Indirect Source Addressing, *R					
1	0110	WS	R	Get source address from WR	
1	0001	SOP	R	Fetch source operand	Ns = 2
Workspace Register Indirect Autoincrement Source Address, *R +					
1	0110	WS	R	Get source address from WR	
1	1001	AUMS			
1	0110	WS	W	Increment WR contents	Ns = 4
1	0001	SOP	R	Fetch source operand	
Symbolic (Direct) Source Address, @LABEL					
1	0010	IOP	R	Get source operand address	
1	0001	SOP	R	Fetch source operand	Ns = 2
Indexed Source Address, @TABLE(R)					
1	0110	WS	R	Fetch base address from WR	
1	0010	IOP	R	Fetch index	
1	1001	AUMS			Ns = 4
1	0001	SOP	R	Fetch source operand	
Workspace Register Destination Address, R					
1	0110	WS	R	Get dest'n operand from WR	Nd = 1
Workspace Register Indirect Destination Address, *R					
1	0110	WS	R	Get dest'n address from WR	
1	0100	DOP		Fetch dest'n operand	Nd = 2
Workspace Register Indirect Autoincrement Destination Address, *R +					
1	0110	WS	R	Get dest'n address from WR	
1	1001	AUMS			
1	0110	WS	W	Increment contents of WR	Nd = 4
1	0100	DOP	R	Fetch dest'n operand	
Symbolic (Direct) Destination Address, @LABEL					
1	0010	IOP	R	Fetch dest'n address	
1	0100	DOP	R	Fetch dest'n operand	Nd = 2
Indexed Destination Address, @TABLE(R)					
1	0110	WS	R	Fetch base address from WR	
1	0010	IOP	R	Fetch index	
1	1001	AUMS			Nd = 4
1	0100	DOP	R	Fetch dest'n operand	

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
A, AB, MOVB, S, SB, SOC, SOCB, SZC, SZCB (See MOV sequence below)				
Ns		<SRC>		Fetch source operand
Nd		<DST>		Fetch dest'n operand
1	0011	IAQ	R	Prefetch next instruction
1	0100*	DOP*	R	Save result at dest'n address
MOV (move word)				
Ns		<SRC>		Fetch source operand
Nd – 1		<DST>		Get dest'n address, but block fetch†
1	0011	IAQ	R	Prefetch next instruction†
1	0100*	DOP*	R	Write operand to dest'n address
SLA, SRA, SRC, SRL (if shift count is NOT zero)				
1	0110	WS	R	Fetch source operand from WR
2	1001	AUMS		Series of 2 consecutive AUMS cycles
CNT‡	1001	AUMS		Repeat shift operation
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	Save result in source WR
SLA, SRA, SRC, SRL (if shift count is zero)				
1	0110	WS	R	Fetch source operand from WR
1	1001	AUMS		
1	0110	WS	R	Fetch shift count from WR0
2	1001	AUMS		Series of 2 consecutive AUMS cycles
CNT‡	1001	AUMS		Repeat shift operation
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Save result in source WR
ABS (source operand in workspace register)				
1	0110	WS	R	Fetch source operand from WR
1	1001	AUMS		
1	0110	WS	R	Save result in source address
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
ABS (non-workspace source operand)				
Ns – 1		<SRC>		Develop address of source operand
1	0000	SOPL	R	Fetch source operand (MPILCK active)
1	1000	AUMSL		
1	0001	SOP	R	Save result in source address
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
AI, ANDI, ORI				
1	0110	WS	R	Fetch source operand from WR
1	0010	IOP	R	Fetch immediate operand
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Save result in source WR

*Substitute WS bus status code if operand is in workspace register.

†The last state of the destination operand derivation sequence is replaced by an instruction fetch.

‡Number of cycles is equal to shift count.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
C, CB				
Ns		<SRC>		Fetch source operand
Nd		<DST>		Fetch dest'n operand
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
CI				
1	0110	WS	R	Fetch source operand from WR
1	0010	IOP	R	Fetch immediate operand
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
AM, SM (double-word add and subtract)				
1	0010	IOP	R	Fetch second word of instruction
1	1001	AUMS		
Ns		<SRC>		Fetch MSW of source operand
Nd – 1		<DST>		Develop destination address†
1	1001	AUMS		Operand fetch is blocked
1	1001	AUMS		
1	0001*	SOP*	R	Fetch LSW of source operand
1	0100*	DOP*	R	Fetch MSW of dest'n operand
1	1001	AUMS		
1	0100*	DOP*	W	Write LSW of result to dest'n address
1	0100*	DOP*	R	Fetch MSW of dest'n operand
1	0011	IAQ	R	Prefetch next instruction
1	0100*	DOP*	W	Write MSW of result to dest'n address
B				
Ns – 1		<SRC>		Get source address†
1	1001	AUMS		No fetch of source operand
1	0011	IAQ	R	Prefetch next instruction from effective source address
1	1001	AUMS		
BIND				
Ns		<SRC>		Fetch source operand
1	1001	AUMS		
1	0011	IAQ	R	Prefetch next instruction from effective source address
1	1001	AUMS		
BL				
Ns		<SRC>		Fetch source operand
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Save old PC in WR11

*Replace with WS bus status code if operand is in workspace registers.

†Block the read cycle in the last cycle of the source fetch sequence.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
BLSK				
1	0110	WS		Fetch stack pointer from WR
1	1001	AUMS		
1	0110	WS	W	Decrement stack pointer in WR
1	0010	IOP	R	Fetch branch address
1	0001	SOP	W	Push return PC onto stack
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
BLWP				
Ns		<SRC>		Fetch source operand (the new WP)
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0001*	SOP*	R	Fetch new PC
1	1100	WP		New WP is output on address lines
1	0110	WS	W	Save old WP in WR13
1	0110	WS	W	Save old PC in WR14
1	0110	WS	W	Save old ST in WR15
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
CLR, SETO				
Ns – 1		SRC>		Get source address†
1	1001	AUMS		No fetch of source operand
1	0011	IAQ	R	Prefetch next instruction
1	0001*	SOP*	W	Save result in source address
COC, CZC				
Ns		<SRC>		Fetch source operand
1	0110	WS	R	Fetch dest'n from designated WR
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
DIV				
Ns		<SRC>		Fetch source operand
1	1001	AUMS		
1	0110	WS	R	Fetch MSW of dest'n operand from WR
IF OVERFLOW, GO TO LABEL 1				Check for divide by zero
1	0110	WS	R	Fetch LSW of dest'n operand from WR + 1
4	1001	AUMS		Series of 4 consecutive AUMS cycles
IF OVERFLOW, GO TO LABEL 1				Is divisor <MSW of dividend?
18	1001	AUMS		Series of 18 consecutive AUMS cycles
1	0110	WS	W	Save quotient in dest'n WR
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Save remainder in dest'n WR + 1
LABEL 1: (GO HERE IF OVERFLOW)				
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		

*Substitute WS bus status code if operand is in workspace register.

†Block the read in the last cycle of the source fetch sequence.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
DIVS				
Ns		<SRC>		Fetch source operand
1	1001	AUMS		
1	0110	WS	R	Fetch LSW of dest'n operand from WR1
1	1001	AUMS		
1	0110	WS	R	Fetch MSW of dest'n operand from WR0
3	1001	AUMS		Series of 3 consecutive AUMS cycles
IF OVERFLOW, GO TO LABEL 1				Check for divide by zero
3	1001	AUMS		Series of 3 consecutive AUMS cycles
IF OVERFLOW, GO TO LABEL 1				Is |divisor| < |dividend|?
20	1001	AUMS		Series of 20 consecutive AUMS cycles
IF OVERFLOW, GO TO LABEL 1				Does unsigned quotient overflow its 15-bit boundary? If so, set ST4.
1	0110	WS	W	Save quotient in WR0
1	0011	IAQ	R	Prefetch next instruction
1	0110	WS	W	Save remainder in WR1
LABEL 1: (GO HERE IF OVERFLOW)				
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
DEC, DECT, INC, INCT, INV, NEG, SWAPB				
Ns		<SRC>		Fetch source operand
1	0011	IAQ	R	Prefetch next instruction
1	0001*	SOP*	R	Save result in source address
LREX, CKOF, CKON, RSET (external instructions)				
4	1001	AUMS		Series of 4 consecutive AUMS cycles
2	1011	I/O	W	I/O cycle is minimum 2 states long
1	1101	ST		Output new status on address bus
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
IDLE (external instruction)				
3	1001	AUMS		Series of 3 consecutive AUMS cycles
2	1011	I/O	W	I/O cycle is minimum 2 clocks long
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
LDCR (parallel load CRU)				
Ns		<SRC>		Fetch source operand
1	0110	WS	R	Get CRU base address from WR12
2	1001	AUMS		Series of 2 consecutive AUMS cycles
2	1001	I/O	W	I/O cycle is minimum 2 states long
1	0011	IAQ	R	Fetch next instruction
1	1001‡	AUMS‡		Increment WR12 if necessary

*Substitute WS bus status code if operand is in workspace register.

‡Substitute WS bus status code and a write cycle if WR12 is post-incremented by 2.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
LDCR (serial load CRU)				
Ns		<SRC>		Fetch source operand
1	0110	WS	R	Fetch CRU base address from WR12
4	1001	AUMS		Series of 4 consecutive AUMS cycles
2* CNT†	1011	I/O	W	I/O cycle is minimum 2 states long
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
LDD AND LDS				
1	1001	AUMS		
1	1001	AUMS		Update internal LDD and LDS flags
1	1001	AUMS		
				MID trap follows
LI				
1	0010	IOP	R	Fetch immediate operand
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	Save operand in specified WR
LIMI				
1	0010	IOP	R	Fetch immediate operand
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0011	IAQ	R	Prefetch next instruction*
1	1001	AUMS		
LST				
1	0110	WS	R	Fetch operand from WR
3	1001	AUMS		Series of 3 consecutive AUMS cycles
1	1101	ST		Output new status on address bus
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
LWP				
1	0110	WS	R	Fetch operand from WR
1	0011	IAQ	R	Fetch next instruction
1	1100	WP		Output new WP on address bus
LWPI				
1	0010	IOP	R	Fetch immediate operand
1	0011	IAQ	R	Fetch next instruction
1	1100	WP		Output new WP on address bus
MPY				
Ns		<SRC>		Fetch source operand
1	0110	WS	R	Fetch dest'n operand from WR
18	1001	AUMS		Series of 18 consecutive AUMS cycles
1	0110	WS	W	Save MSW of result in WR
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	Save LSW of result in WR + 1

†The number of cycles is specified in the count field of the opcode.

*The new mask controls interrupts.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
MPYS				
Ns		<SRC>		Fetch source operand
1	1001	AUMS		
1	0110	WS	R	Fetch dest'n operand from WR
19	1001	AUMS		Series of 19 consecutive AUMS cycles
1	0110	WS	W	Save MSW of result in WR
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	Save LSW of result in WR + 1
RTWP (return from subroutine in main memory)				
1	1001	AUMS		
1	0110	WS	R	Fetch new PC from WR14
1	0110	WS	R	Fetch new ST from WR15
1	0110	WS	R	Fetch new WP from WR13
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	1101	ST		Output new ST on address bus
1	0011	IAQ	R	Prefetch next instruction
1	1100	WP		Output new WP on address bus
RTWP (return from using opcodes >380, >382, or >384				
5	1001	AUMS		Series of 5 consecutive AUMS cycles
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		
Jump Instructions				
1	1001	AUMS		
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
SLAM, SRAM				
1	0010	IOP	R	Fetch second word of opcode
1	1001	AUMS		
Ns		<SRC>		Fetch MSW of source operand
IF SHIFT COUNT IS ZERO, GO TO LABEL 1				
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0001*	SOP*	R	Fetch LSW of source operand
2	1001	AUMS		Series of 2 consecutive AUMS cycles
CNT†	1001	AUMS		Repeat shift operation
1	0001*	SOP*	W	Save LSW of source operand
1	0011	IAQ	R	Fetch next instruction
1	0001*	SOP*	W	Save MSW of source operand
LABEL 1: (GO HERE IF SHIFT COUNT IS ZERO)				
1	1001	AUMS		
1	0110	WS	R	Fetch shift count in WR0
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0001*	SOP*	R	Read LSW of source operand
1	1001	AUMS		
IF SHIFT COUNT IN R0 IS ZERO, GO TO LABEL 2				
1	1001	AUMS		
CNT†	1001†	AUMS†		Repeat shift operation until done
1	0001*	SOP*	W	Write LSW of result to source address
1	0011	IAQ	R	Fetch next instruction
1	0001*	SOP*	W	Write MSW of result to source address
LABEL 2: (GO HERE IF SHIFT COUNT IN WR0 IS ZERO)				
1	1001	AUMS		
1	0001*	SOP*	W	Write LSW of result to source address
1	0011	IAQ	R	Prefetch next instruction
1	0001*	SOP*	W	Write MSW of result to source address

*Substitute WS bus status code if operand is in workspace register.

†Number of cycles is equal to shift count.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
STCR (parallel store CRU)				
Ns†				Fetch source operand if byte transfer
NOTE: SOURCE OPERAND IS NOT FETCHED IF WORD TRANSFER				
1	0110	WS	R	Read I/O base address from WR12
2	1001	AUMS		Series of 2 consecutive AUMS cycles
2	1011	I/O	R	I/O cycle is minimum 2 states long
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0001†	SOP†	W	Save result in source address
1	0011	IAQ	R	Prefetch next instruction
1	1001†	NOP†		Increment WR12 if necessary
STCR (bit-serial store CRU)				
Ns‡				Fetch source operand if byte transfer
1	0110	WS	R	Fetch I/O base address from WR12
5	1001	AUMS		Series of 5 consecutive AUMS cycles
2§CNT*	1011	I/O	R	I/O read takes min. 2 states/bit
3	1001	AUMS		Series of 3 consecutive AUMS cycles
IF 8 OR 16 BITS TRANSFERRED, GO TO LABEL 1				
?	1001	AUMS		Repeat cycle 8-N for byte or 16-N for word, where N = number of bits
LABEL 1:				
1	0011	IAQ	R	Fetch next instruction
1	0001§	SOP§	W	Save result in source address
SBO, SBZ (single-bit CRU instructions)				
1	1001	AUMS		
1	0110	WS	R	Fetch I/O base address from WR12
1	1001	AUMS		
2	1011	I/O	W	I/O cycle is minimum 2 states long
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		

*Number of cycles is equal to count field from STCR opcode.

†Substitute WS bus status code if WR12 is post-incremented by 2.

‡If source operand is word rather than byte, fetch of operand is replaced by AUMS cycle.

§Substitute WS bus status code if operand is in workspace register.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
TB				
1	1001	AUMS		
1	0110	WS	R	Fetch I/O base address from WR12
1	1001	AUMS		
2	1011	I/O	R	I/O cycle is minimum 2 states long
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
TMB, TCMB, TSMB (source operand in workspace register)				
1	0010	IOP	R	Fetch second word of instruction
1	0110	WS	R	Fetch source operand from WR
2	1001	AUMS		Series of 2 consecutive AUMS cycles
Bit displacement†	1001	AUMS		Shift target bit into position
2	1001	AUMS		Series of 2 consecutive AUMS cycles
16-bit displacement‡	1001	AUMS		Restore shifted bit to original position
1	1001	AUMS		
1	0110	WS	W	Write result to WR
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
TMB, TCMB, TSMB (non-register source operand)				
1	0010	IOP	R	Fetch second word of instruction
Ns – 1		<SRC>		Get source address (see next cycle)
1	0000	SOPL	R	Fetch source with MPILCK active
2	1000	AUMSL		Series of 2 consecutive AUMSL cycles
Bit displacement†	1000	AUMSL		Shift target bit into position
2	1000	AUMSL		Series of 2 consecutive AUMSL cycles
16-bit displacement‡	1000	AUMSL		Restore shifted bit to original position
1	1000	AUMSL		
1	0001	SOP	W	Save results and deactivate MPILCK
1	0011	IAQ	R	Prefetch next instruction
1	1001	AUMS		

†Number of cycles is equal to the bit number plus one.

‡Number of cycles is equal to 16 minus the bit number.

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONTINUED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
X				
Ns		<SRC>		Fetch source operand (target opcode)
1	1001	AUMS		
				Execute target opcode
STST, STWP				
1	1001	AUMS		
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	Save result in WR
XOP				
Ns – 1		<SRC>		Get source operand address (see next)
1	1001	AUMS		Block fetch of source operand
1	1101	ST		Output all zeros on address bus
1	1001	AUMS		
1	0101	INTA	R	Fetch new WP from vector
1	1001	AUMS		
1	1100	WP		Output new WP on address bus
1	1001	AUMS		
1	0110	WS	W	Save source address in WR11
1	0101	INTA	R	Fetch new PC from vector
1	0110	WS	W	Save old WP in WR13
1	0110	WS	W	Save old PC in WR14
1	0110	WS	W	Save old ST in WR15
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
XOR				
Ns		<SRC>		Fetch source operand
1	0110	WS	R	Fetch dest'n operand from WR
1	0011	IAQ	R	Fetch next instruction
1	0110	WS	W	

TABLE 19 – INSTRUCTION EXECUTION SEQUENCES (CONCLUDED)

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
EVAD (This instruction is available only in Macrostore)				
Ns		<SRC>*		Fetch source operand
1	1001	AUMS		
1	1001	AUMS		Save Macrostore PC in WR4 of Macrostore
1	1001	AUMS		Fetch user's PC from WR14 of Macrostore
2	1001	AUMS		Series of 2 consecutive AUMS cycles
IF TARGET OPCODE SOURCE ADDRESS IS *R+, GO TO LABEL 1				
Ns – 1		<SRC>		Get source address for target word
GO TO LABEL 2				
LABEL 1:				
1	0110	WS	R	Fetch source address from user's WR
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	1001	AUMS		Save address of user's WR in WR10
1	1001	AUMS		
LABEL 2:				
IF TARGET OPCODE DESTINATION ADDRESS IS *R+, GO TO LABEL 3				
Nd – 1		<DST>		Get dest'n address for target word
GO TO LABEL 4				
LABEL 3:				
1	0110	WS	R	Fetch dest'n address from user's WR
1	1001	AUMS		Save address of user's WR in WR9
LABEL 4:				
3	1001	AUMS		Series of 3 consecutive AUMS cycles
1	1001	AUMS		Save updated user PC in WR14 of Macrostore
1	1001	AUMS		Restore Macrostore PC
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	1001	AUMS		Save dest'n address in WR7 of Macrostore
1	1001	AUMS		Fetch next instruction
1	1001	AUMS		Save source address in WR8 of Macrostore

*All cycles output AUMS bus status code.

TABLE 20 – INTERRUPT AND MACROSTORE TRAP SEQUENCES

NUMBER	BUS		READ	
OF	STATUS		OR	
CYCLES	CODE	NAME	WRITE	COMMENT
INTERRUPTS				
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	1101	ST		Output all zeros on address bus
1	0101	INTA	R	Fetch new WP from interrupt vector
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	0101	INTA	R	Fetch new PC from interrupt vector
1	1100	WP		Output new WP on address bus
1	0110	WS	W	Save old WP in WR13
1	0110	WS	W	Save old PC in WR14
1	0110	WS	W	Save old ST in WR15
1	0011	IAQ	R	Fetch next instruction
1	1001	AUMS		
TRAP TO MACROSTORE (MID trap)				
1	1001	AUMS		
1	1110	MID		Check for attached processor
1	1001	AUMS		
1	1001	AUMS		Save contents of main IR in WR5
1	1001	AUMS		
1	1001	AUMS		If MID trap is due to 2nd word of instruction, save PC-2 in WR14
1	1001	AUMS		Save LDS and LDD flags and first word of 32-bit opcode in WR3
1	1001	AUMS		Read Macrostore PC from vector
1	1001	AUMS		
1	1001	AUMS		Save user's WP in WR13
1	1001	AUMS		Save user's PC in WR14
1	1001	AUMS		Save user's ST in WR15
2	1001	AUMS		Series of 2 consecutive AUMS cycles

TABLE 21 – ATTACHED PROCESSOR INTERFACE SEQUENCES

NUMBER OF CYCLES	BUS STATUS CODE	NAME	READ OR WRITE	COMMENT
99000 TRANSFERS CONTROL TO ATTACHED PROCESSOR (MID trap)				
1	1001	AUMS		
1	1110	MID		Check for attached processor
2	1001	AUMS		Series of 2 consecutive AUMS cycles
1	1101	ST		Output all zeros on address bus
1	0101	INTA	R	Fetch WP from level-2 vector
3	1001	AUMS		Series of 3 consecutive AUMS cycles
1	1100	WP		Output new WP on address bus
1	0110	WS	W	Save old WP in WR13
1	0110	WS	W	Save old PC in WR14
1	0110	WS	W	Save old ST in WR15
1	1111	HOLDA		Release bus to attached processor
ATTACHED PROCESSOR RETURNS CONTROL TO 99000				
1	1111	HOLDA		Last state of hold cycle
1	0110	WS	R	Fetch new PC from WR14
1	0110	WS	R	Fetch new ST from WR15
1	0110	WS	R	Fetch new WP from WR13
1	1101	ST		Output new ST on address bus
1	0011	IAQ	R	Fetch next instruction
1	1100	WP		Output new WP on address bus

11. TMS99105A/TMS99110A PRELIMINARY ELECTRICAL SPECIFICATIONS

11.1 ABSOLUTE MAXIMUM RATINGS OVER OPERATING FREE-AIR TEMPERATURE RANGE (UNLESS OTHERWISE NOTED)†

Supply voltage, V_{CC} (see Note 1)	−0.3 to 7 V
All input voltages	−0.3 to 20 V
Output voltages	−0.3 to 7 V
Continuous power dissipation	1000 mW
Operating free-air temperature	0 °C to 70 °C

† Stresses beyond those listed under"Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Note 1: All voltage values are with respect to V_{SS}

11.2 RECOMMENDED OPERATING CONDITIONS

PARAMETER	MIN	NOM	MAX	UNITS
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except CLKIN)	2		$V_{CC}+1$	V
High-level input voltage, V_{IH} (CLKIN)	3.5		$V_{CC}+1$	V
Low-level input voltage, V_{IL} (all inputs except CLKIN)	−1		0.8	V
Low-level input voltage, V_{IL} (CLKIN)			0.2	V
High-level output current, I_{OH} (All outputs)			400	µA
Low-level output current, I_{OL} (all outputs)			2‡	mA
Operating free-air temperature, T_A	0		70	°C

‡Output current of 2 mA is sufficient to drive 5 low-power Schottky TTL loads or 10 advanced low-power Schottky TTL loads (worst case).

11.3 ELECTRICAL CHARACTERISTICS OVER RECOMMENDED FREE-AIR TEMPERATURE (UNLESS OTHERWISE NOTED)

PARAMETER		TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{OH}	High-level output voltage	V_{CC}=MIN,	I_{OL}=MAX	2.4			V
V_{OL}	Low-level output voltage all except BST(1-3), R/$\overline{W}$, $\overline{MEM}$	V_{CC}=MIN,	I_{OL}=MAX			0.5	V
V_{OL}	Low-level output voltage, (BST(1-3), R/$\overline{W}$, $\overline{MEM}$	V_{CC}=MIN,	I_{OL}=MAX			0.6	V
I_O	Tristate (high-impedance) output current (off)	V_{CC}=MAX	V_O=2.4 V			20	µA
			V_O=0.4 V			−20	
I_I	Input current	V_I=V_{SS} to V_{CC}				20	µA
I_{CC}	Supply current	V_{CC}=MAX			120		mA
C_I	Input capacitance (all inputs except address/data lines)	f = 1 MHz, all other pins at 0 V			15		pF
C_{DB}	Address/data line capacitance				25		pF
C_O	Output capacitance (except address/data lines)				10	15	pF

†For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡All typical values are at V_{CC} = 5 V, T_A = 25°C.

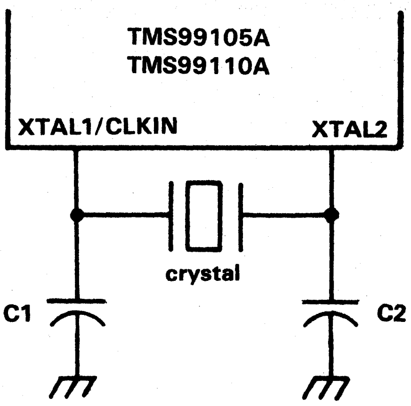
11.4 CLOCK CHARACTERISTICS

The TMS99000 has an internal oscillator and 4-phase clock generator controlled by an external crystal or resistor-capacitor combination. Alternatively, the user can directly inject a frequency source into the XTAL1 input. The period of the frequency source must be one-fourth the desired machine state time.

11.4.1 Internal Oscillator

The internal oscillator is enabled by connecting a crystal across XTAL1 and XTAL2. The machine state time, t_s, is four times the crystal oscillator period, $1/f_x$. The crystal should be a fundamental series-resonant type. Figure 33 presents the circuit configuration for this mode of operation.

ADVANCE INFORMATION
This document contains information on a new product. Specifications are subject to change without notice.



NOTES: 1. The crystal should be a fundamental series-resonant type operating at four times the machine state frequency.
2. C1 and C2 represent the total capacitance on these pins, including strays and parasitics.

FIGURE 33 – INTERNAL OSCILLATOR

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Crystal frequency, f_x	0-70 °C		24		MHz
C1, C2	0-70 °C		5		pF

11.4.2 External Clock

An external clock of frequency f_x may be connected to the XTAL1/CLKIN in place of a crystal or RC combination. The period of the CLKOUT output signal will be $4/f_x$. Figure 34 shows the circuit configuration when an external clock is used.

PARAMETER		MIN	NOM	MAX	UNIT
f_{ext}	External source frequency	12		24	MHz
$t_{c\phi}$	CLKIN cycle time	41.25		83.33	ns
$t_{r\phi}$	CLKIN rise time (see Note 1)		4	10	ns
$t_{f\phi}$	CLKIN fall time (see Note 1)		4	10	ns
t_{wH1}	CLKIN high-level pulse width		$t_{c\phi}/2$-$t_{r\phi}$		ns
t_{wL1}	CLKIN low-level pulse width		$t_{c\phi}/2$-$t_{f\phi}$		ns

Note 1: CLKIN rise and fall times are a function of V_{IH} and V_{IL}. For the times shown the V_{IH} and V_{IL} levels are as given under "Recommended Operating Conditions." If a maximum 5 ns rise and fall time can be achieved, then the V_{IH} and V_{IL} levels may be standard levels of 2.4 V and 0.4 V respectively.

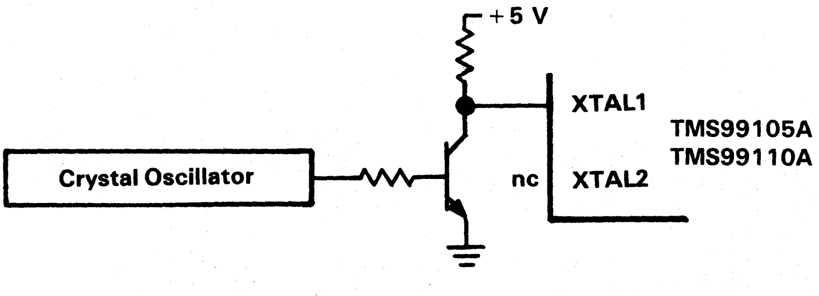


FIGURE 34 – EXTERNAL OSCILLATOR

ADVANCE INFORMATION
This document contains information on a new product. Specifications are subject to change without notice.

11.5 TIMING REQUIREMENTS OVER RECOMMENDED OPERATING CONDITIONS

PARAMETER		MIN	NOM	MAX	UNIT
t_{su1}	READY setup time prior to falling edge of CLKOUT	35			ns
t_{h1}	READY hold time after falling edge of CLKOUT	0			ns
t_{su2}	Data setup time prior to falling edge of CLKOUT	30			ns
t_{h2}	Data hold time after falling edge of CLKOUT	0			ns
t_{su3}	$\overline{\text{INTREQ}}$, $\overline{\text{RESET}}$, $\overline{\text{APP}}$ setup time prior to falling edge of CLKOUT	40			ns
t_{su4}	$\overline{\text{HOLD}}$ setup time prior to falling edge of CLKOUT	80			ns
t_{h3}	$\overline{\text{INTREQ}}$, $\overline{\text{RESET}}$, $\overline{\text{APP}}$, $\overline{\text{HOLD}}$ hold time after falling edge of CLKOUT	15			ns
t_{acc}	Access time, address valid to data valid at data setup time	$3t_{c2}/4$-40			ns
t_{ded}	$\overline{\text{RD}}$ low until valid data required	$t_{c2}/2$-63			ns

11.6 SWITCHING CHARACTERISTICS OVER RECOMMENDED OPERATING CONDITIONS

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{c2}	CLKOUT cycle time (f(x) = crystal freq)	C_L = 100 pF (See Figure 35)		$4t_{c1}$ or $4/f_x$		ns
t_{r2}	CLKOUT rise time			10	15	ns
t_{f2}	CLKOUT fall time			10	15	ns
t_{wH2}	CLKOUT high-level pulse width			$t_{c2}/2$-t_{r2}		ns
t_{wL2}	CLKOUT low-level pulse width			$t_{c2}/2$-t_{f2}		ns
t_{wH3}	ALATCH pulse width high			$t_{c2}/4$-t_{r2}		ns
t_{d1}	Delay time, reference line to ALATCH low				$t_{c2}/4+13$	ns
t_{d2}	Delay time, ref line to ALATCH high			11	15	ns
t_{d3}	Delay time, ref line to $\overline{\text{MEM}}$, BST, R/$\overline{\text{W}}$ address, $\overline{\text{PSEL}}$ valid			13	15	ns
t_{d4}	Delay time, ALATCH low to address, $\overline{\text{PSEL}}$ invalid		10	15	20	ns
t_{dz1}	Delay time, ALATCH to address hi-z		20	30	35	ns
t_{d5}	Delay time, ref line to start of $\overline{\text{WE}}$ invalid		$t_{c2}/4$			ns
t_{d6}	Delay time, ALATCH low to start of $\overline{\text{RD}}$ invalid		10	24	30	ns
t_{d7}	Delay time, CLKOUT low to $\overline{\text{WE}}$, $\overline{\text{RD}}$ high			15	30	ns
t_{d8}	Delay time, ALATCH low to data valid		20	30	35	ns
t_{d9}	Delay time, ref line to $\overline{\text{WE}}$ valid			$t_{c2}/4+13$	$t_{c2}/4+20$	ns
t_{d10}	Delay time, $\overline{\text{WE/IOCLK}}$ high to data invalid		$t_{c2}/4-30$			ns
t_{d11}	Delay time, CLKOUT low to data, $\overline{\text{PSEL}}$, BST, $\overline{\text{MEM}}$, R/$\overline{\text{W}}$ invalid		$t_{c2}/4+5$			ns
t_{d12}	Delay time CLKOUT low to $\overline{\text{WE/IOCLK}}$, $\overline{\text{RD}}$ inactive		0			ns
t_{drde}	Delay time ref line to $\overline{\text{RD}}$ low				$t_{c2}/4+40$	ns
t_{dz2}	Delay time, ref line to R/$\overline{\text{W}}$ hi-z				$t_{c2}/4+25$	ns
t_{dz3}	Delay time, ref line to ALATCH hi-z				$t_{c2}/4+25$	ns
$t_{d(rav)}$	Delay time, ref line to ALATCH invalid		$t_{c2}/4$			ns
t_{d13}	Delay time, ALATCH low to address, $\overline{\text{MEM}}$, BST, $\overline{\text{RD}}$, $\overline{\text{WE}}$, R/$\overline{\text{W}}$ invalid		10			ns
t_{d14}	Delay time ref line to address, $\overline{\text{MEM}}$, BST, $\overline{\text{RD}}$, $\overline{\text{WR}}$, R/$\overline{\text{W}}$ invalid			$t_{c2}/4+20$		ns

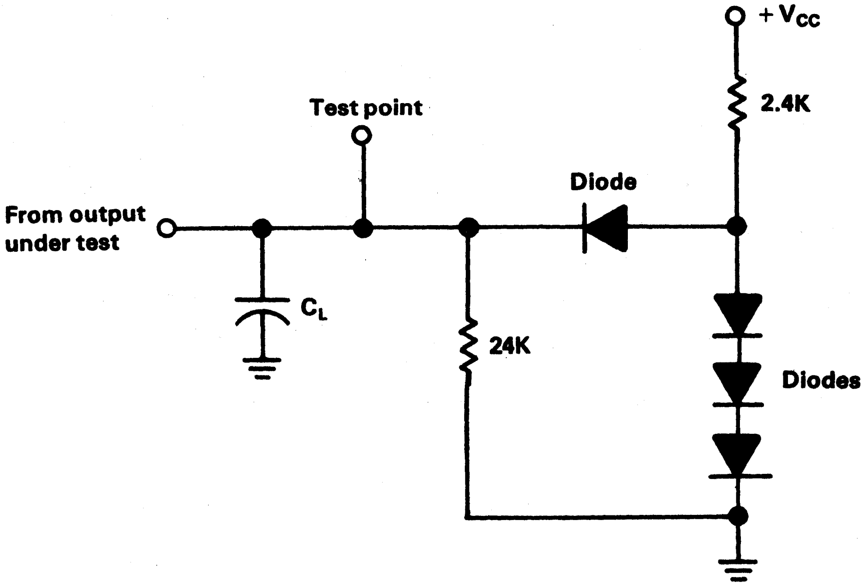
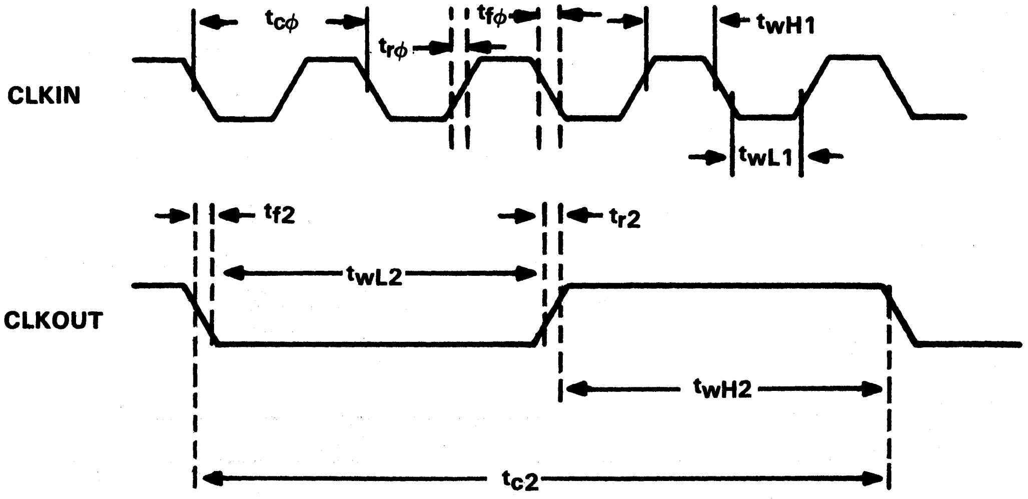


FIGURE 35 – SWITCHING TIMES LOAD CIRCUIT



NOTES: (1) There is no time relationship implied or specified between the input clock and the output clock.
(2) All timing reference points are 10% and 90% points.

FIGURE 36 – CLOCK TIMING PARAMETERS

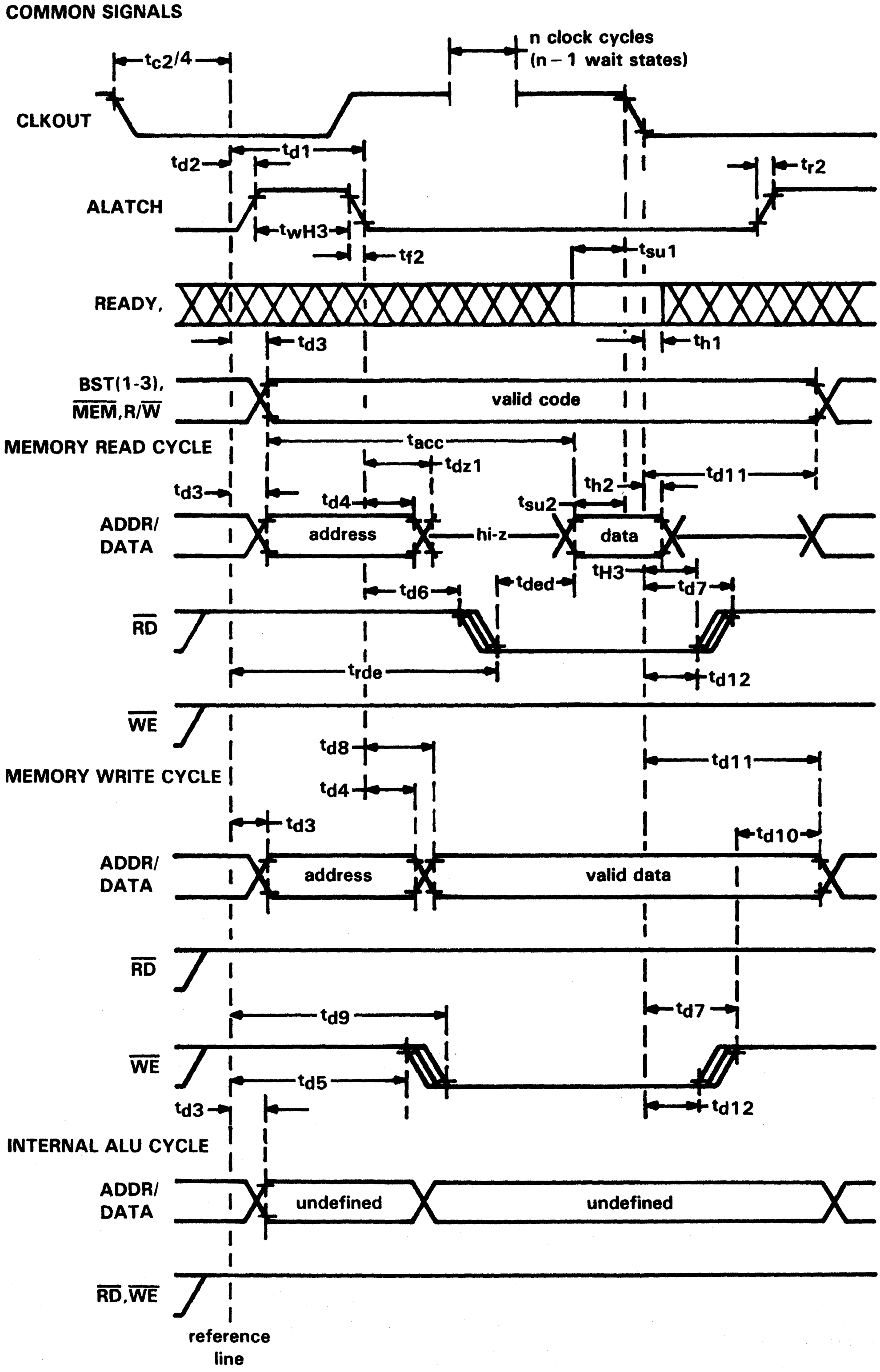


FIGURE 37 – MEMORY AND INTERNAL CYCLE TIMING PARAMETERS

All timing reference points are 10% and 90% points.

COMMON SIGNALS

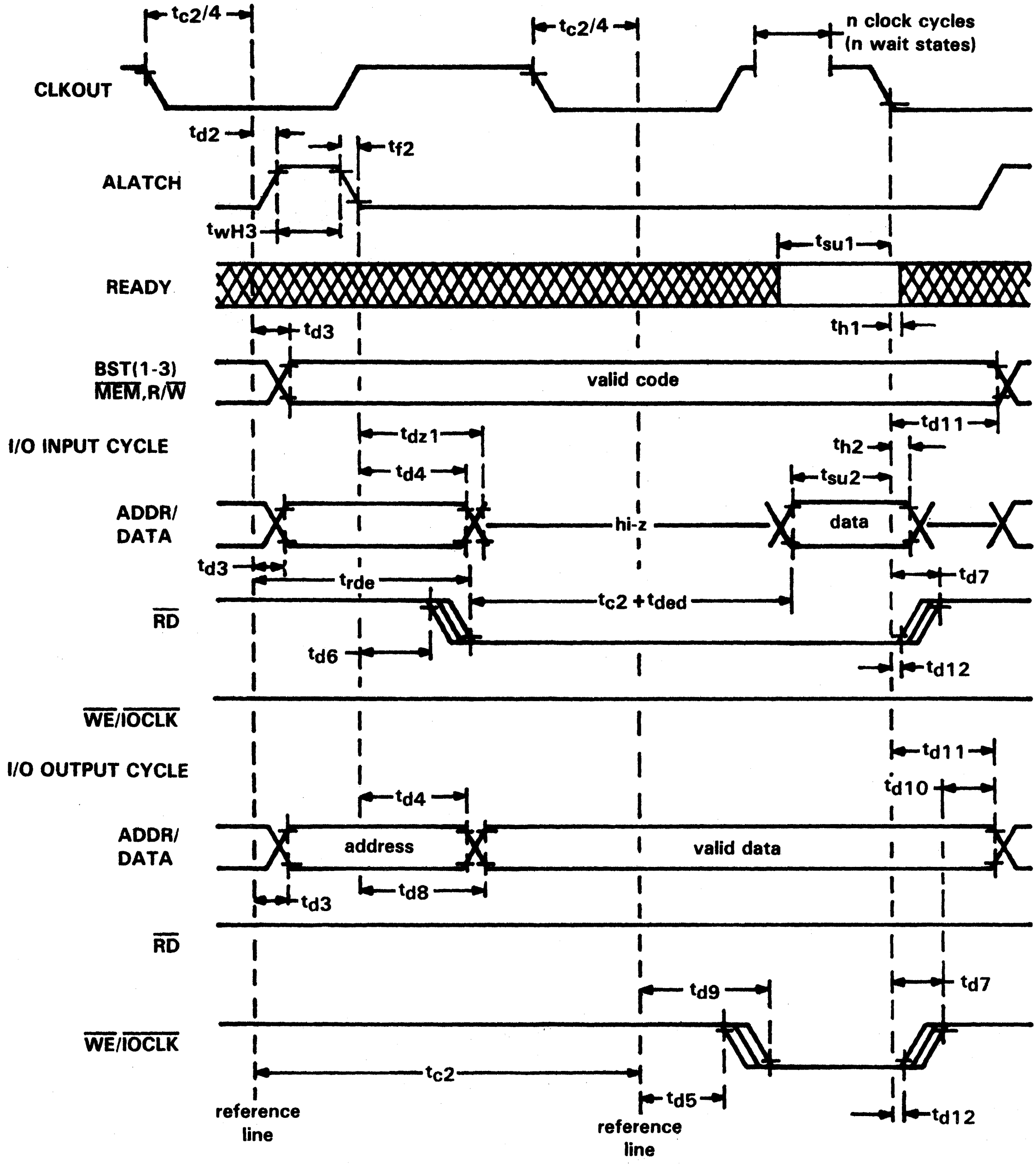
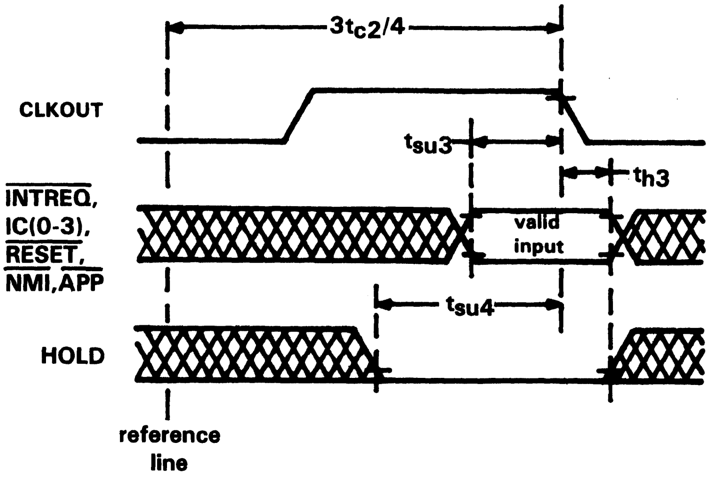


FIGURE 38 – I/O CYCLE TIMING PARAMETERS

All timing reference points are 10% and 90% points.



NOTE: The CLKOUT edge at which each of the input signals is sampled is defined in the section pertaining to that signal.

FIGURE 39 – INTERRUPT, HOLD AND APP TIMING PARAMETERS

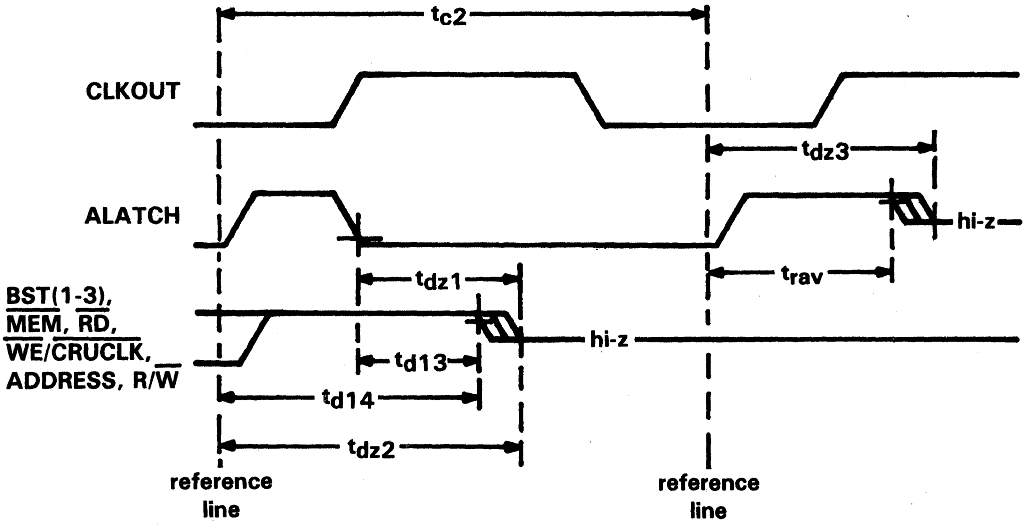


FIGURE 40 – HOLD CYCLE TIMING PARAMETERS

All timing reference points are 10% and 90% points.

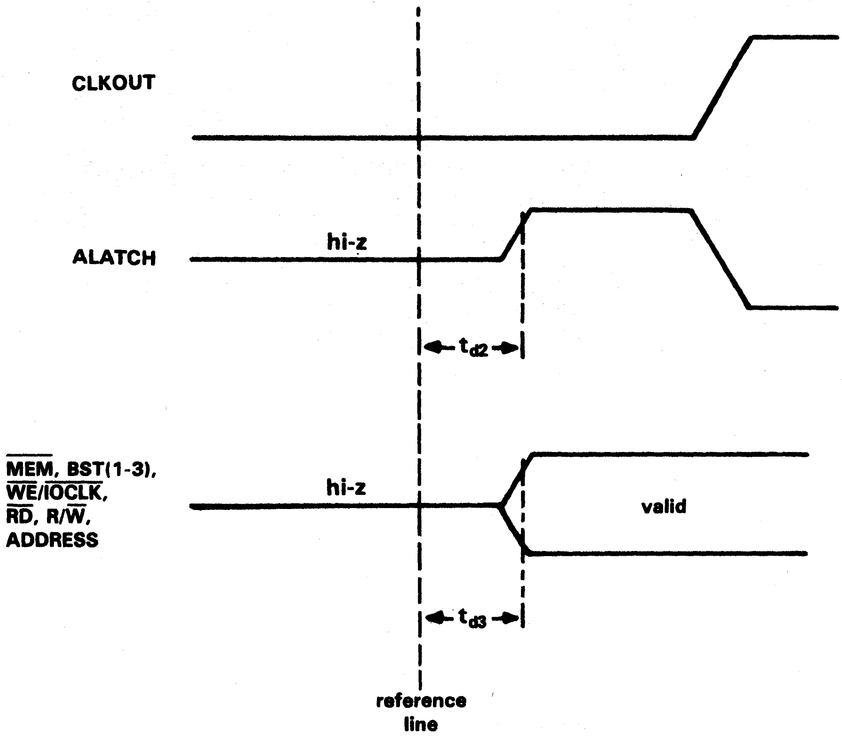
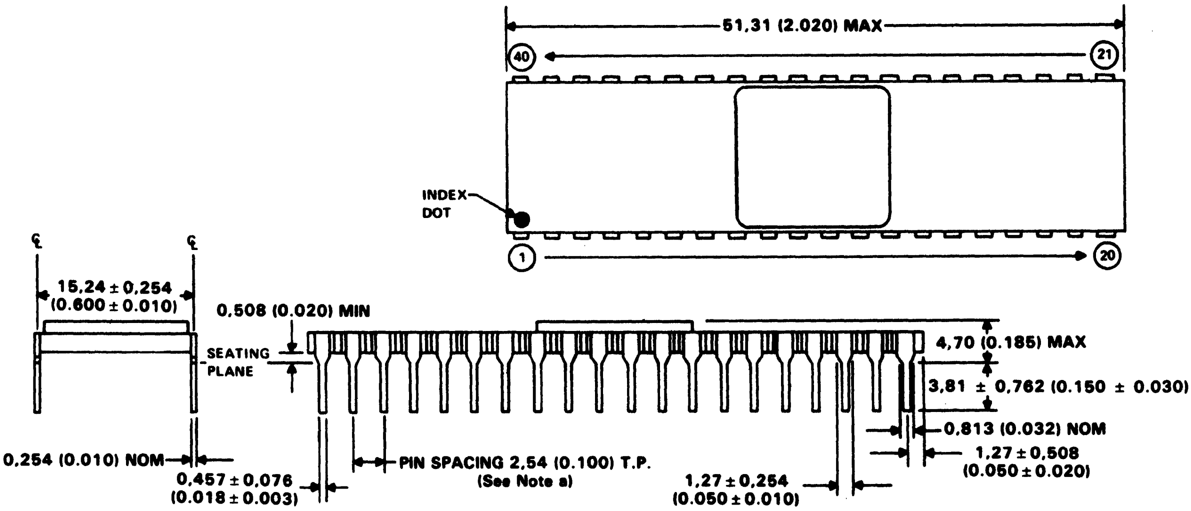

CLKOUT
ALATCH
hi-z
t_{d2}
$\overline{\text{MEM}}$, BST(1-3),
$\overline{\text{WE}}/\overline{\text{IOCLK}}$,
$\overline{\text{RD}}$, R/$\overline{\text{W}}$,
ADDRESS
hi-z
valid
t_{d3}
reference line

FIGURE 40 – $\overline{\text{HOLD}}$ CYCLE TIMING PARAMETERS (CONCLUDED)

All timing reference points are 10% and 90% points.

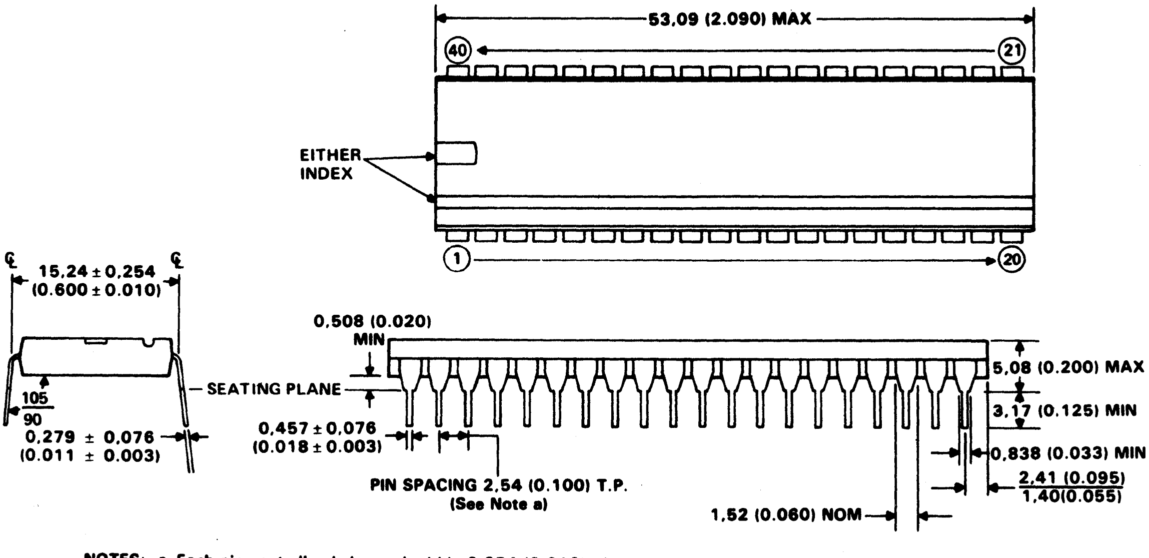
12. TMS99000 MECHANICAL SPECIFICATIONS

12.1 TMS99105A/TMS99110A – 40-PIN CERAMIC PACKAGE



NOTES: a. Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.
b. All linear dimensions are in millimeters and parenthetically in inches. Inch dimensions govern.

12.2 TMS99105A/TMS99110A – 40-PIN PLASTIC PACKAGE



NOTES: a. Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.
b. All linear dimensions are in millimeters and parenthetically in inches. Inch dimensions govern.

APPENDIX A
TMS99105A SUPPLEMENT

A. TMS99105A DESCRIPTION

The TMS99105A is the basic member of the TMS99000 microprocessor family. The TMS99105A offers the same features as described in Sections 1 through Section 12. The only feature not implemented on the TMS99105A is on-chip Macrostore. However, external Macrostore may be utilized for user-implemented functions and instructions.

A.1 TMS99105A MACROSTORE OPERATION

The TMS99105A may utilize external Macrostore by placing the TMS99105A in prototyping mode (see Section 7.2.4.2). If no external Macrostore is to be implemented in the system, it is recommended that the TMS99105A be placed in baseline mode (Section 7.2.4.3). If no external Macrostore is implemented on the TMS99105A and the standard mode or prototyping mode is selected, the occurrence of a MID opcode will result in indeterminate operation. This is due to the fact that a Macrostore vector will occur to non-existent Macrostore memory (see Section 7.3) and potentially cause a system lockup.

APPENDIX B
TMS99110A SUPPLEMENT

B TMS99110A DESCRIPTION

The TMS99110A 16-bit microprocessor is a powerful member of the TMS99000 family that implements 12 single precision floating point instructions, 2 memory mapper control instructions and a 32 X 32 signed integer multiply instruction. These powerful instruction set enhancements are implemented via the on-chip Macrostore memory space. They are additions to the instruction set described in Section 10.

B.1 TMS99110A MACROSTORE OPERATION

With the instruction set extensions of the TMS99110A implemented in Macrostore, it is required that the TMS99110A be generated in standard mode (see Section 7.2.4.1). If either the prototyping mode or baseline mode of operation is selected, the instruction set extensions described in this section will not be operational.

B.2 TMS99110A INSTRUCTION SET EXTENSION SUMMARY

The TMS99110A implements the instructions listed Table B.1 in addition to those listed in Section 10. Note that these instructions are operational only when the TMS99110A is operated in the standard mode.

TABLE B.1 – TMS99110A INSTRUCTION SET EXTENSION SUMMARY

MNEMONIC	DESCRIPTION	OPERATION*
LDD	Long Distance Source	Update internal LDD flag
LDS	Long Distance Destination	Update internal LDS flag
AR	Add Real	FPAC+(SA,SA+2) → FPAC
SR	Subtract Real	FPAC − (SA,SA+2) → FPAC
MR	Multiply Real	(SA,SA+2) * FPAC → FPAC
DR	Divide Real	FPAC / (SA,SA+2) → FPAC
LR	Load Real	(SA,SA+2) → FPAC
STR	Store Real	FPAC → (SA,SA+2)
NEGR	Negate Real	− FPAC → FPAC
CR	Compare Real	(SA,SA+2) − (DA,DA+2) set status
CIR	Convert Integer to Real	Convert (SA) → FPAC
CER	Convert Extended Integer to Real	Convert FPAC → FPAC
CRI	Convert Real to Integer	Convert FPAC to integer → FPAC
CRE	Convert Real to Extended Integer	Convert FPAC to ex. integer → FPAC
MM†	Multiply Multiple (32×32)	(SA,SA+2) * (DA,DA+2) → (DA,DA−2,DA+4,DA+6

*Floating point accumulator (FPAC) is designated as workspace registers 0 and 1 of the current workspace.

†MM is not a floating point operation but is an addition to the TMS99110A instruction set.

B.3 TMS99110A MEMORY MAPPER CONTROL INSTRUCTIONS

The LDD and LDS instructions are provided for use in controlling a 16-register memory map file. These instructions are implemented on the 99110 only.

These mapper instructions are intended to support the use of the TIM99610 (SN74LS610) memory mapper (see the SN74LS610 data sheet).

General Format:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	OPCODE										0	0	0	0	0	0

MNEMONIC	OPCODE 0123 4567 89	MEANING
LDS	0000 0111 10	Long Distance Source
LDD	0000 0111 11	Long Distance Destination

The LDD and LDS instructions permit data to be accessed through the user's memory map while in the supervisor mode. The $\overline{\text{PSEL}}$ pin will be inverted during the source (if LDS) or destination (if LDD) operand access cycles of the following instruction, unless the addressing mode is register direct. When register direct addressing ($T_S = 0$ or $T_D = 0$) is used for the source or destination operand fetch, $\overline{\text{PSEL}}$ is not altered.

Listed below are the effects when an LDD or LDS instruction follows an LDS or LDD instruction. In general, only the first two cases, LDD followed by LDS or LDS followed by LDD, are considered useful:

LDD—LDS: Both the source and destination memory cycles of the instruction that follows are long distance.
LDS—LDD: Same as LDD—LDS.
LDS – LDS: The first LDS has no effect; the source memory cycles of the instruction that follow are long distance.
LDD – LDD: The first LDD has no effect; the destination memory cycles of the instruction that follow are long distance.

Interrupts are inhibited until after the next instruction.

An attempt to execute a long-distance instruction while in user mode (ST7 = 1) will be flagged as a privileged opcode violation.

The LDS or LDD instruction has no effect if the source or destination addressing mode of the target instruction is workspace register direct.

If the instruction to be long distanced is interruptible, the long distance information can be recovered upon return from the interrupt. An example of an interruptible instruction is a block move with multiple source and destination operands to which long distancing is applied. Typically, a checkpoint or loop count register keeps track of the number of moves completed. If long distancing is not applied to either operand, the normal procedure, when an interrupt occurs, is to store the loop count and other pertinent instruction status in a checkpoint register and decrement the program counter. After the interrupt is serviced, execution continues from where it stopped. After the instruction is completed, the check point register is set to – 1 or some other value to indicate that the instruction will be executed for the first time when it is next encountered.

Several features have been incorporated into the LDD and LDS instructions to facilitate recovery from an interrupted instruction when an LDD and/or an LDD instruction is active. Both the LDD and the LDS instructions save the address plus 2 (main memory) of the first LDD or LDS in a possible sequence of LDD s and/or LDS s. Any Macrostore implemented instructions, which could be long distanced and interrupted, must not accidentally destroy this data.

The three most significant bits of Macrostore location >0006 (WR3 if WP = 0) are long distance status information as shown in the following table. The fact that a long distance instruction is active may be determined by comparing the contents of >0006 to the value > E000. If the comparison is greater than or equal, then the long distance instruction is not active.

LONG-DISTANCE FLAGS 0 1 2	MEANING
1 1 1	No long distancing in effect
1 1 0	Long distance source
1 0 1	Will never occur
1 0 0	Long distance source*
0 1 1	Long distance destination
0 1 0	Sequence: 1) LDD, 2) LDS (order is significant)
0 0 1	Will never occur
0 0 0	Sequence: 1) LDS, 2) LDD, or see Note.†

* A sequence of two LDS instructions has been encountered. If emulating the 990/12 version of LDS, the source operand access of the second LDS is controlled by the first LDS to be long distance.

† The sequence LDS, LDD, LDS has been encountered. If emulating the 990/12 version of LDD and LDS, then the source operand access of the LDD is controlled by the first LDS to be long distance.

The procedure for handling an interruptible instruction is relatively simple due to the information stored in locations > 0004 and >0006 (WR2 and WR3 if WP = 0). When an interrupt is detected by using one of the jump on interrupt instructions, first do all necessary clean-up (such as updating the checkpoint register(s)), and compare Macrostore location >0006 to value > E000 to determine if a long distance instruction was active. If no long distance is active, then load the contents of WR14 (used to return back to main memory) with the address of the start of the instruction. It may be necessary to save the contents of WR14 on entering Macrostore for this purpose because WR14 may be modified by executing the Macrostore routine or by an EVAD instruction. If a long distance is active, then the contents of location >0004 must be decremented by 2 and then loaded into WR14 so that after returning with an RTWP (> 0380), the PC will point to the start of the string of LDDs and/or LDSs. Since an interrupt caused the Macrostore routine to be exited, the interrupt vector will be taken immediately upon return to main memory via the RTWP instruction. Upon returning back to Macrostore from the interrupt(s), the PC will be initialized with the value at the time of Macrostore exit thus restarting the Macrostore routine.

The long-distance flags are automatically cleared if the exit from Macrostore is performed by executing the >0380 or >0382 form of RTWP but are NOT cleared if >0384 is executed. The long-distance flags are also automatically cleared after the instruction following the LDD or LDS has been completed. Note that the long-distance flags have no effect on the $\overline{\text{PSEL}}$ output during Macrostore accesses. $\overline{\text{PSEL}}$ always represents the complemented value of ST8 when executing out of Macrostore memory space. Note if an SOP or DOP bus status code is output while in Macrostore, it will cause $\overline{\text{PSEL}}$ to flip if the corresponding LDD or LDS is active.

B.4 TMS99110A FLOATING POINT INSTRUCTIONS

The floating point package of the TMS99110A provides floating point operations. The general method is to load the Floating Point Accumulator (FPAC – R0,R1 of user's workspace) with one operand, perform the desired operation, and then store the result found in the FPAC (see examples below). The floating point instructions are only available to the TMS99110A when the processor is initialized in standard mode (Section 7.2.4.1). When in prototyping mode, the execution of these opcodes will cause a trap to external Macrostore memory space for user defined opcodes. The user should avoid the use of these opcodes to prevent possible conflicts with future TMS99110A floating point capability. When in baseline mode, execution of these opcodes will cause a level 2 illegal opcode interrupt. The following is the general format of a floating point number:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sgn	Exponent							1st Digit				2nd Digit			
3rd Digit				4th Digit				5th Digit				6th Digit			

Where:

Sgn → Sign of the number, 0 = Pos, 1 = Neg

Exponent → Exponent (radix 16) of the number + >40 (e.g., for the mantissa to be raised to the 2nd power, exponent would be 2 + >40 = >42)

Digits 1-6 → Mantissa of the number (in hex).

NOTE: The TMS99110A assumes the decimal point place to be prior to the first digit of the mantissa. It also assumes that the number is a floating poi number and not zero, the first digit is non-zero. If a floating zero is to be represented, the exponent should also be cleared (set to 00). Otherwis errors could result.

TABLE B.2 – TMS99110A FLOATING POINT FUNCTIONS

AR	Add Real	to FPAC
SR	Subtract Real	to FPAC
MR	Multiply Real	to FPAC
DR	Divide Real	to FPAC
LR	Load Real	into FPAC
STR	Store Real	from FPAC
NEGR	Negate Real	in FPAC
CR	Compare Reals	general source/dest
CIR	Convert Integer to Real	general source to FPAC
CER	Convert Extended Integer to Real	in FPAC
CRI	Convert Real to Integer	in FPAC
CRE	Convert Real to Extended Integer	in FPAC
MM	Multiply Multiple (32 bit Integer × 32 bit Integer = >64 bi general source/destination	

To perform a floating point function, the package uses R0 and R1 of the user's workspace as a floating point accumulator. All floating point operations (except for MM and CR) use the FPAC. To load the accumulator use the LR instruction or manually move the desired operand into R0-R1. To store a result, the STR instruction could be used or the number could be manually moved out (see example below).

EXAMPLE 1—ALTERNATE METHODS OF A SIMPLE OPERATION

LR *R4	Load FPAC	MOV *R4,R0 MOV @2(R4),R0
AR R5	Do Add Read	AR R5
STR @ANS	Store Answer	MOV R0,@ANS MOV R1,@ANS+2

EXAMPLE 2—A MORE INVOLVED FLOATING POINT SEQUENCE

Suppose the following equation was to be evaluated and a 'lowest value calculated' parameter replaced if the result was even smaller.

$$\frac{\text{V1} * (-\text{V2} - \text{CONSTANT})}{\text{V3} * (\text{V4} + 2.)}$$

Assuming the parameters were already off in memory some place, the following would be a possible solution. (Note addressing modes)

```
        LI      R0,2            LOAD INTEGER 2 INTO FRAC               (hi word only)
        CIR     R0              CONVERT IT TO REAL                     (register direct)
        AR      *R2             ADD DENOMINATOR TERM V4                (indirect)
        MR      *R3+            MULTIPLY DENOMINATOR TERM V3           (indirect auto-inc)
        STR     R8              STORE TEMP RESULT                      (register direct)
*
        LR      @CONST          GET CONSTANT                           (symbolic)
        CER                     CONVERT EXTENDED INTEGER TO REAL       (FPAC content)
        NEGR                    NEGATE FPAC CONTENTS
        SR      @OFFSET(R4)     SUBTRACT NUMERATOR TERM V2             (indexed)
        MR      *R5             MULTIPLY NUMERATOR TERM V1             (indirect)
        DR      R8              DO THE DIVISION                        (indirect)
        CR      R0,@LOW         COMPARE VS LOWEST                      (direct & symbolic)
        JGT     LOOP            JUMP IF NOT LOWER (OR EQUAL)
        STR     @LOW            STORE NEW LOWEST                       (symbolic)
LOOP    ••• •••  (etc.etc.etc.)
```

B.4.1 Dual-Operand Floating Point Instructions with Multiple Addressing Modes for the Source Operand (99110A only)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
General Format:	0	0	0	0	OPCODE						Ts		S			

The addressing mode for the source operand is determined by the Ts field.

Ts	S	ADDRESSING MODE	NOTES
0	0,1,. . . .,15	Workspace register	
1	0,1,. . . .,15	Workspace register indirect	
2	0	Symbolic	
2	1,2,. . . .,15	Indexed	1
3	1,2,. . . .,15	Workspace register indirect autoincrement	1,2

NOTES: 1. Workspace register 0 may not be used.
2. The workspace register is incremented by 4.

MNEMONIC	OPCODE 456789	MEANING	RESULT COMPARED TO 0	STATUS BITS AFFECTED	DESCRIPTION
AR	110001	Add Real	Yes	0-4	FPAC + (SA,SA+2) → FPAC
SR	110011	Subtract Real	Yes	0-4	FPAC − (SA,SA+2) → FPAC
MR	110100	Multiply Real	Yes	0-4	(SA,SA+2) * FPAC → FPAC
DR	110101	Divide Real	Yes	0-4	FPAC / (SA,SA+2) → FPAC
LR	110110	Load Real	Yes	0-2	(SA,SA+2) → FPAC
STR	110111	Store Real	Yes	0-2	FPAC → (SA,SA+2)
CIR*	110010	Convert Int to Real	Yes	0-4	Real Representation of (SA) → FPAC

*CIR is actually a single operand function; however, its operand is pointed to by SA, not necessarily the FPAC.

B.4.2 Single-Operand Floating Point Instructions

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	0	0	0	0	1	1	0	0	0	0	0	0	0	OPCD		0

	OPCODE		RESULT	STATUS	
	11		COMPARED	BITS	
MNEMONIC	34	MEANING	TO 0	AFFECTED	DESCRIPTION
CRI	00	Convert Real to Int	Yes	0-4	Int Representation of FPAC → FPAC
NEGR	01	Negate Real	Yes	0-2	– FPAC → FPAC
CRE	10	Convert Real to Ext Int	Yes	0-4	Ext Int Representation of FPAC → FPAC
CER	11	Convert Ext Int to Real	Yes	0-4	Real Representation of FPAC → FPAC

B.4.3 Dual-Operand Floating Point Instructions with Multiple Addressing Modes for the Source and Destination Operands

General	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Format:	0	0	0	0	0	0	1	1	0	0	0	0	OPCODE			
	0	0	0	0	Td		D				Ts		S			

The addressing mode for the operands is determined by the Tx fields (X being either D or S).

Tx	X	ADDRESSING MODE	NOTES
0	0,1, . . . ,15	Workspace register	
1	0,1, . . . ,15	Workspace register indirect	
2	0	Symbolic	
2	1,2, . . . ,15	Indexed	1
3	1,2, . . . ,15	Workspace register indirect autoincrement	1,2

NOTES: 1. Workspace register 0 may not be used.

2. The workspace register is incremented by 4 unless X → D and opcode → 0010 (MM) in which case the workspace register is incremented by 8.

	OPCODE		RESULT	STATUS	
	1111		COMPARED	BITS	
MNEMONIC	2345	MEANING	TO 0	AFFECTED	DESCRIPTION
CR	0001	Compare Reals	No	0-4	(SA,SA+2) – (DA,DA+2) Set Status
MM	0010	Multiply Multiple	Yes	0-4	(SA,SA+2) * (DA,DA+2) → (DA,DA+2,DA+4,DA+6) (Unsigned, Integer)

B.4.4 Status Bit Summary for Floating Point Instructions

The following table summarizes the conditions that set the status register bits during execution of floating point instructions.

TABLE B.3 – ADDITIONS FOR THE 99110 VERSION

BIT	NAME	INSTRUCTION	CONDITION TO SET BIT TO 1 (OTHERWISE SET TO 0)
ST0	Logically greater than	AR,SR,MR, DR,LR,STR, NEGR,CIR, CER,CRI, CRE,CR	If result is not 0
		MM	Cleared unconditionally
ST1	Arithmetic greater than	AR,SR,MR DR,LR,STR NEGR,CIR, CER,CRI, CRE	If MSB of result = 0, and result is not 0
		CR	If (SA) > (DA)
		MM	Cleared unconditionally
ST2	Equal/TB Indicator	AR,SR,MR DR,LR,STR, NEGR,CIR, CER,CRI, CRE,MM	If result = 0
		CR	If (SA) = (DA)
ST3	Carry out	LR,STR,NEGR	Unaffected
		AR,SR,MR,DR	If exponential overflow occurs
		MM,CR	Cleared unconditionally
		CIR,CER	Set unconditionally
		CRI,CRE	If real source cannot be represented if the format selected
ST4	Arithmetic Fault	LR,STR,NEGR	Unaffected
		AR,SR,MR,DR	If exponential over/underflow occurs
		MM,CR, CIR,CER	Cleared unconditionally
		CRI,CRE	If real source cannot be represented if the format selected
ST5-ST15		All Floating Point Instructions	Unaffected

TEXAS INSTRUMENTS

November 1982
MP009

Post Office Box 1443   Houston, Texas 77001
Semiconductor Group

Printed in U.S.A.