CHAPTER 4

Hardware Design: Architecture and Interfacing Techniques

INTRODUCTION

Describing the 9900 system from a hardware standpoint clearly requires detailed descriptions of a large number of design features as well as the interaction between the 9900 and peripheral circuits. In this chapter, material is arranged to develop a 9900 system from the viewpoint of the 9900 microprocessor chip. In the architecture section, the concepts of instruction fetch and decode, the memory-to-microprocessor bus structures, and memory partitioning (the use of volatile and non-volatile memories) are explained. Other topics include descriptions of the registers on the microprocessor chip and the working registers, the concept of memory-to-memory architecture, timing and descriptions of interface signals.

A special section covers memory in detail, especially the controls and timing, multichip memory structure, static and dynamic RAM, and DMA (direct memory access).

Following the architecture and memory sections are sections devoted to the instruction set, design considerations for input/output techniques especially in CRU development, the interrupt structure and electrical requirements.

A special section devoted to the unique features of the single chip microcomputer, the TMS9940, is included at the end of the chapter.

Information in this chapter flows from the most basic fundamentals to an understanding of the more complex design features of the 9900 and the chip family. When very specific and detailed information regarding pin assignments and speed is given, the TMS9900 device specifications are used. These examples will give direction and illustration for interpreting the data sheet information found in Chapter 8.

The 9900 family of 16-bit microprocessors includes several device types each aimed at a specific market segment. The same basic architecture and instruction set are maintained throughout. Consider first the single-chip microprocessor which consists of an ALU (arithmetic and logic unit), a few registers, and instruction handling circuitry (*Figure 4-1*). There is no memory on the chip for instructions and data so it must be interfaced to memory devices, usually RAM for data (and instructions which must be modified) and ROM, PROM, or EPROM for instructions (*Figure 4-2*). It is often desirable to store instructions in a non-volatile memory to eliminate the requirement for loading the program into memory immediately following application of power. This is especially important in dedicated applications where the program is fixed and power off-on cycles are common occurrences.

The microprocessor is connected to memory devices and external input/output (I/O) devices via sets of signals or busses (*Figure 4-2*). An address bus selects a word of memory. The contents of this word will be transferred to or from the microprocessor via the data bus. Control signals required to effect the transfer of information between the microprocessor and the memory are grouped into a control bus.

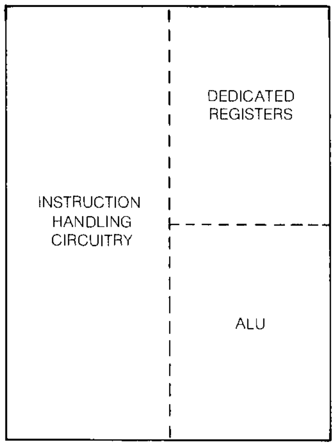


Figure 4-1. The 9900 Microprocessor

The interface to external devices (I/O) may be accomplished by using the address, data and control busses. This technique is known as parallel I/O or memory mapped I/O because data is transferred in parallel and the I/O devices occupy locations in the memory address space.

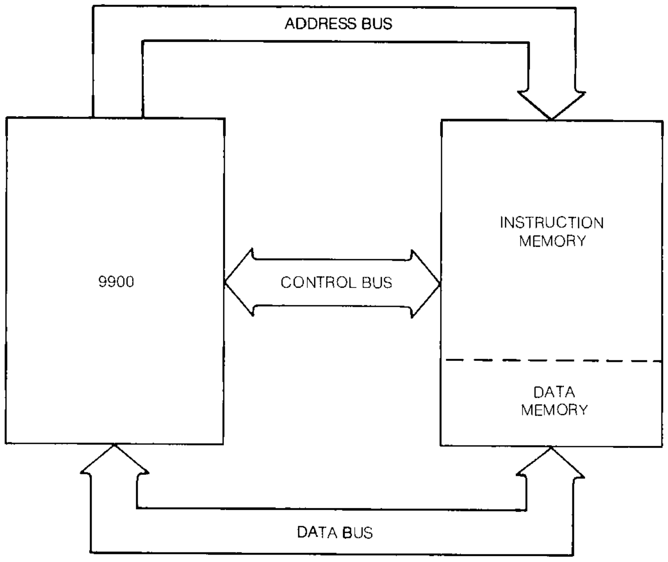


Figure 4-2. 9900 Microprocessor and Memory

The extension of parallel I/O is direct memory access (DMA). External hardware is employed to act as a separate special purpose processor for transferring large blocks of contiguous memory words to or from an external device (such as a disc memory). Once such a transfer is set up (via a string of instructions in the program), the DMA controller automatically synchronizes the transfer of data between the external device and memory, sharing the buses timewise with the microprocessor.

The 9900 architecture includes one other important I/O technique. Designed primarily for single bit I/O transfers, the communications register unit (CRU) provides a powerful alternative to parallel, memory mapped I/O (*Figure 4-3*). The address bus is used to select one of 4096 individual input or output bits in the CRU address space. During the execution of one of the single bit CRU instructions, the processor transfers one bit in or out. Multiple bit instructions are also available which provide for transfer of up to sixteen bits via a single CRU operation.

▶4 While this chapter describes primarily the basic TMS9900 16-bit microprocessor, all of the 9900 family CPU's are covered in detail in the *Product Data* chapter.

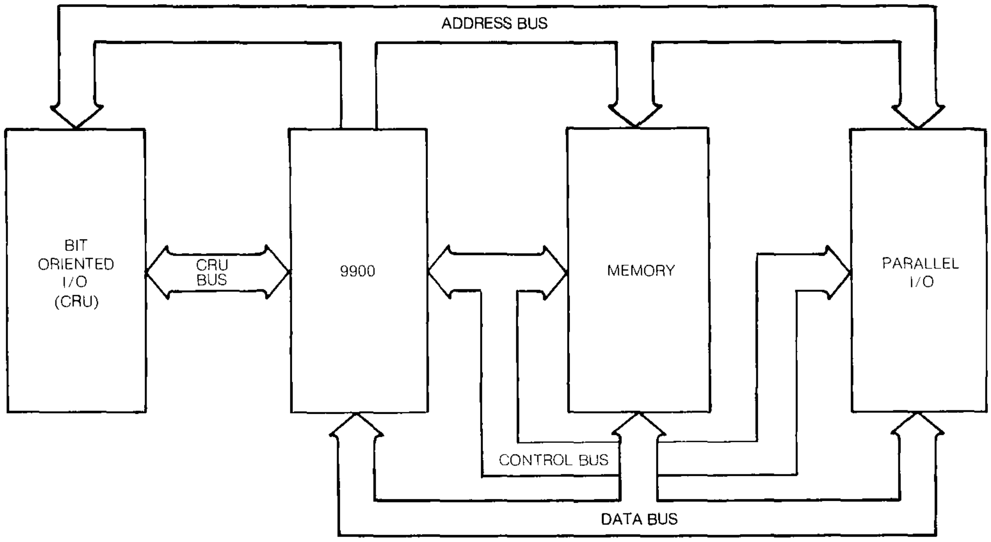


Figure 4-3. *9900 Bus Architecture*

An overview is given here to establish design paths for microprocessor systems. Listed below are the processors in the 9900 family.

Device	*Technology*	*Description*
TMS 9900	N-MOS	16-bit CPU 3 MHz
TMS 9900-40	N-MOS	16-bit CPU 4 MHz
SBP 9900A	I^2L	16-bit CPU −55° to 125°C
TMS 9980A/81	N-MOS	16-bit CPU 40-pin package
TMS 9985	N-MOS	16-bit CPU 40-pin package
TMS 9940	N-MOS	16-bit CPU with 2 k on-chip ROM

General purpose applications are designed around the TMS9900 device. The same is true for systems with severe environmental specs; however, a transition to the SBP9900A is made after the design is complete and the software completely debugged. The TMS9980A/81 and the TMS9985 are used where the 40-pin package is advantageous and a slightly slower speed is acceptable. The TMS9940 is a single-chip microcomputer for small special purpose controllers.

At the end of this chapter and in the *Product Data* chapter there is detailed design data for application of the LSI (large scale integration) peripheral support circuits in the 9900 family which are available for use in 9900 microprocessor-based systems. But in order to read and understand the data presented in this chapter and in this book, an understanding of the basic fundamentals of microprocessors is needed.

ARCHITECTURE

Basic Microprocessor Chip

The 9900 is an advanced 16-bit LSI microprocessor with minicomputer-like architecture and instructions. It is easy to understand and easy to use. Consider first the microprocessor device itself (*Figure 4-4*). Operations are carried out with a set of dedicated registers, an ALU, and instruction handling circuits. As clock signals are applied, the processor will fetch an instruction word from a memory (external to the chip), will execute it, fetch another instruction, execute it and so on. In each case the instruction is saved in an instruction register (IR) on the chip. The decode circuit sets up the appropriate controls based on the content of the instruction register for a multi-step execution phase. A memory address register (MAR) is used to hold address information on the address bus. The ALU and the other registers perform their specified functions during the execute phase of the instruction cycle.

Microprocessor Registers

There are three registers on the 9900 chip which are the key architectural features of the microprocessor (*Figure 4-5*). They are the workspace pointer (WP), the program counter (PC), and the status register (ST).

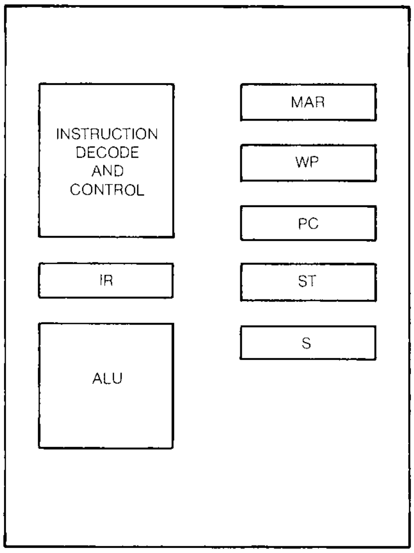


Figure 4-4. 9900 Functional Elements

Workspace Pointer

The general purpose registers for the 9900 are implemented as blocks of memory called workspaces. A workspace consists of 16 contiguous words of memory, but are general registers to the user. The workspace pointer on the 9900 chip holds the address of the first word in the workspace. After initializing the content of the WP at the beginning of a program (or subprogram), the programmer may concentrate on writing a program using the registers to hold data words or to address data elsewhere in memory.

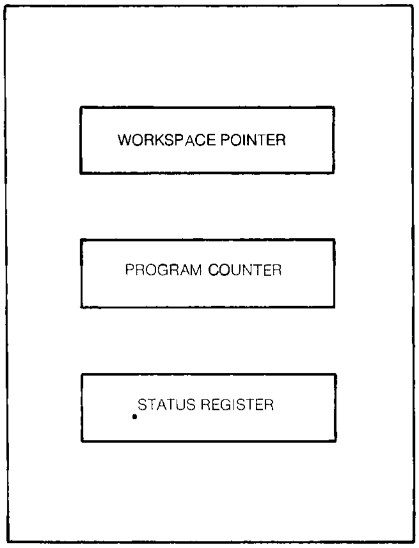


Figure 4-5. Three Important Registers

Program Counter

The program counter (PC) in the 9900 is used in the conventional way to locate the next instruction to be executed. As each instruction is executed, the program counter is incremented to the next consecutive word address. Because word addresses are even numbers in the 9900, the program counter is incremented by two in order to address sequential instructions. If the instruction to be executed occupies two or three memory words, the program counter will be incremented to generate sequential (even) addresses to access the required number of words. At the end of execution the PC is incremented to the next even address which is the location of the next instruction. If the instruction to be executed is a jump or branch instruction, the program counter is loaded with a new address and program execution continues starting with the instruction at that location in memory.

Figure 4-6 shows the program counter pointing to (addressing) instruction words in the program. Starting with location (x) the instructions are performed in sequence until a jump is encountered at (y). Processing resumes sequentially starting at location (z) which was the address specified by the jump instruction to be placed in the program counter.

Status Register

The status register (ST) is the basis for decision making during program execution. Individual bits of the ST are set as flags as the result of instructions. They may thereafter be tested in the execution of conditional jump instructions. *Figure 4-7* shows the status register and its flag bits.

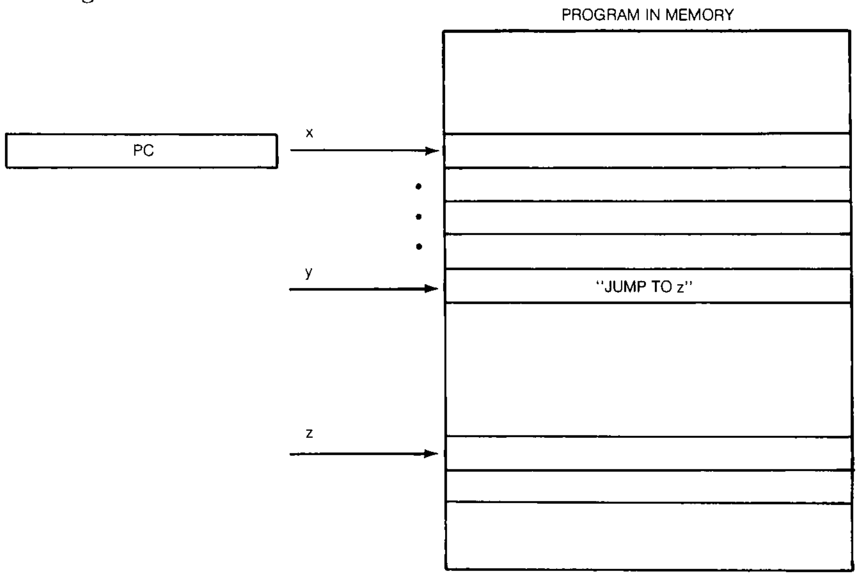


Figure 4-6. *Program Counter Operation*

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
L>	A>	=	C	O	P	X	RESERVED					INTERRUPT MASK			

Bit	Function
0	Logical "Greater Than"
1	Arithmetic "Greater Than"
2	Equal
3	Carry
4	Overflow
5	Parity
6	XOP Instruction Being Executed
12-15	Interrupt Mask

Figure 4-7. Status Register

The first three bits are set as a result of comparisons. Some instructions identify two operands (numbers) to be compared. If the first is greater than the second, the "greater than" bit should be set. In the 9900 there are two such conditions. First, the logical-greater-than bit considers 16-bit words as positive integers and the comparison is made accordingly. Second, the arithmetic-greater-than bit is set as the result of a comparison of two numbers which are considered in two's complement form. For example: consider the numbers A and B below as the numbers in the compare instruction C A, B:

```
A  1000  1110  1100  0101

B  0110  1010  1100  1101
```

If they are 16-bit positive integers, it is clear from the most significant bits (MSB) that A is greater than B, and the logical-greater-than bit of the status register should be set to one. But as two's complement numbers, A is negative (MSB = 1) and B is positive. Therefore the arithmetic-greater-than bit must be made zero (A is not greater than B). Since the processor has no way of knowing how the designer has used the memory words for data (integers or two's complement), two status bits must be provided for decision making. The designer can select the appropriate conditional jump instruction (testing status bit 0 or 1) because he knows what the data format is.

Status bit 2, the equal bit, is set if the two words compared are equal.

In many instructions, only one number is involved or a new number is determined as the result of an arithmetic operation. For these instructions status bits 0, 1 and 2 are set as the result of comparisons against zero; that is, if the single number or answer obtained is greater than zero or equal to zero.

Memory-to-Memory Architecture

The 9900 family of processors employs memory-to-memory architecture in the execution of instructions. Memory-to-memory architecture is that computer organization and instruction set which enables direct modification of memory data via a single instruction. That is, a single instruction can fetch one or two operands from memory, perform an arithmetic or logical operation, and also store the result in memory. In doing so, some of the on-chip registers are used as temporary buffers in much the same manner as an accumulator is used in other systems. But instructions to load an accumulator and store the accumulator are rarely necessary in memory-to-memory architecture. A single 9900 instruction (arithmetic or logical) does the work of two or more instructions in other systems.

Figure 4-8 describes the technique used by the 9900 to locate words in memory as "registers" in the workspace. Additional information is included for reference purposes. Registers 1-15 may be used for indexing (see the description of this addressing mode in Chapter 5 and 6). Register 0 may be used for a shift count. Registers 11 and 13-15 are used for subroutine techniques. Register 12 is a base value for CRU instructions. These special uses of the workspace registers are stated here as an initial evaluation of the register set. Program control and CRU instructions make use of the contents of registers 11-15; therefore, programmers and systems designers must be aware that while use of these registers is not restricted to their special functions, they should be used with caution in performing other functions.

The use of these workspaces in an actual application is best described in the Software Design chapter. But the step-by-step execution of the instructions is of concern in hardware design because of the execution speed and the techniques for handling interrupts.

Instruction cycles in the 9900 require memory access not only for the instruction words but also for operand addresses and actual operands (or numbers to be operated upon.) A simple add instruction requires at least four memory cycles: one to fetch the instruction, two to access the two numbers to be added, and one to store the result. As will be explained in detail later in this chapter, the execution of an add instruction may require as many as eight memory cycles (because of the addressing mode.) The execution steps are not the same for all instructions. There is, in fact, substantial variation of execution steps within any one instruction due to addressing. Tables and charts are provided in this chapter to explain the execution time of each instruction.

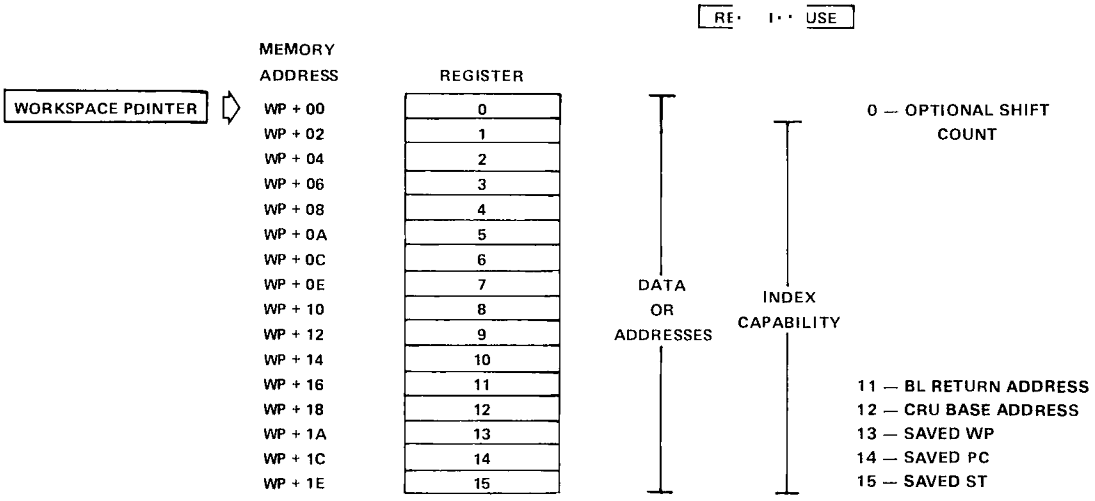


Figure 4-8. 9900 Workspace Registers

There is one additional concept regarding microprocessor and memory interfacing to be introduced at this time: it is the way in which data is stored in the memory. *Figure 4-9* shows the bit numbering for a general 16-bit data word or instruction. Instructions and 16-bit data words are always located at even addresses. Since the memory is byte addressable, even and odd bytes are the left and right half words in the 16-bit memory organization and have even or odd addresses respectively. Memories for the TMS9900 and SBP9900A contain 16 bits per word, while the other processors in the family use 8-bit memory structures. But all use the same addressing concept: a 16-bit address describing a 64k-byte address space.

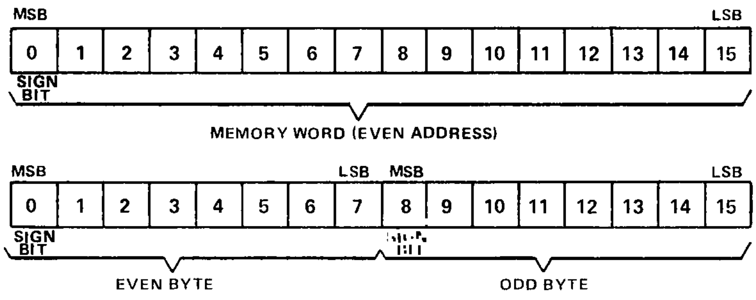


Figure 4-9. Word and Byte Formats

Context Switching

One of the more important advantages of the workspace architecture of the 9900 is the fact that "register save and restore" operations are greatly simplified. In any interrupt processing system, provisions must be made to perform an orderly transition into a new program segment in response to an interrupt. In other microprocessor systems, the first few instructions of an interrupt service routine perform the steps of saving register contents in memory, and then loading new values into the registers.

In the 9900, an interrupt cycle starts with a hardware operation to save the contents of the three key registers, the WP, PC and ST. In addition, the WP and PC must be loaded with new numbers. *Figures 4-10* and *4-11* show an example of the technique. Prior to the interrupt, the WP locates the workspace (pointing to 0800), the PC locates the current instruction (pointing to 0100), and the ST contains the status as a result of the execution of the current instruction (e.g., 4000). At the end of execution, the processor tests for an interrupt condition and finding it, performs a *context switch* as follows.

Step 1. The new WP value is fetched from the appropriate interrupt vector location in the first 32 words of memory. This identifies the location of the workspace assigned to the interrupt service routine.

Step 2. The current values of the WP, PC and ST registers are stored in the new workspace – ST in R15, PC in R14, WP in R13 in that order. After this, the new PC value is fetched from memory (the second location of the two-word interrupt vector) and loaded into the PC.

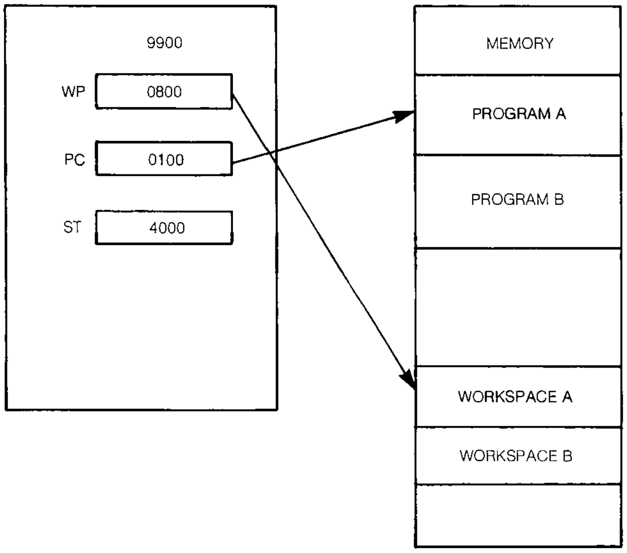


Figure 4-10. Before Context Switch

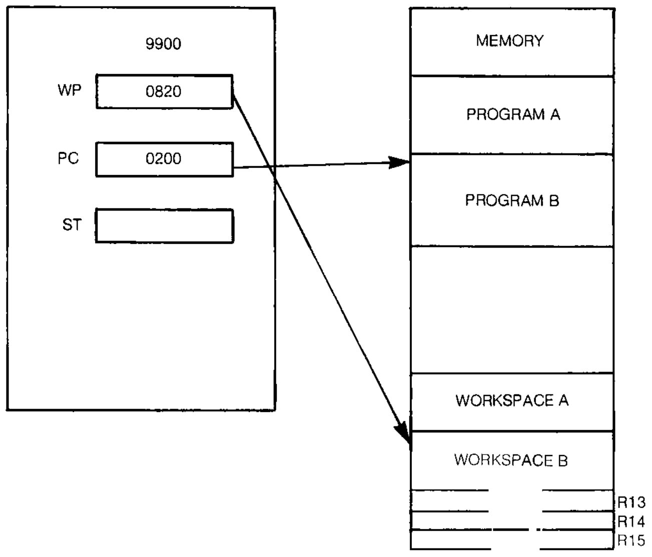


Figure 4-11. After Context Switch

▶4

Step 3. With the context switch completed, processing resumes with the first instruction in the interrupt service routine.

Processing continues in this mode until, at the end of the interrupt routine, an RTWP instruction is encountered. A "reverse" context switch now occurs to return to the previous program. Since R13, 14 and 15 contain the control register contents for the previous program, they are now transferred to the CPU which loads them into the WP, PC and ST. Processing resumes from the point at which the interrupt occurred.

The obvious advantage of context switching is the reduced register-save register-restore operations required by microprocessors in an interrupt environment. The context switch is also used as a subroutine technique. This is described in Chapters 5 and 6, but the important fact is that context switching is, to the designer, a single step, when in fact several steps are performed by the microprocessor.

MEMORY

The 9900 is easily interfaced to any of the standard types of semiconductor memory devices. Texas Instruments provides masked ROMs, field-programmable ROMs (PROMs), and erasable PROMs (EPROMs) for non-volatile program and data storage. RAMs are available in sizes from a 64 x 8 static RAM to the 64K dynamic RAMs for use as a temporary program and data storage. 9900-compatible memory devices are listed in Chapter 2.

Memory Organization

The 9900 instructions build a 16-bit address word which describes a 64K x 8 bit address space. A memory map for the 9900 is shown in *Figure 4-12.*

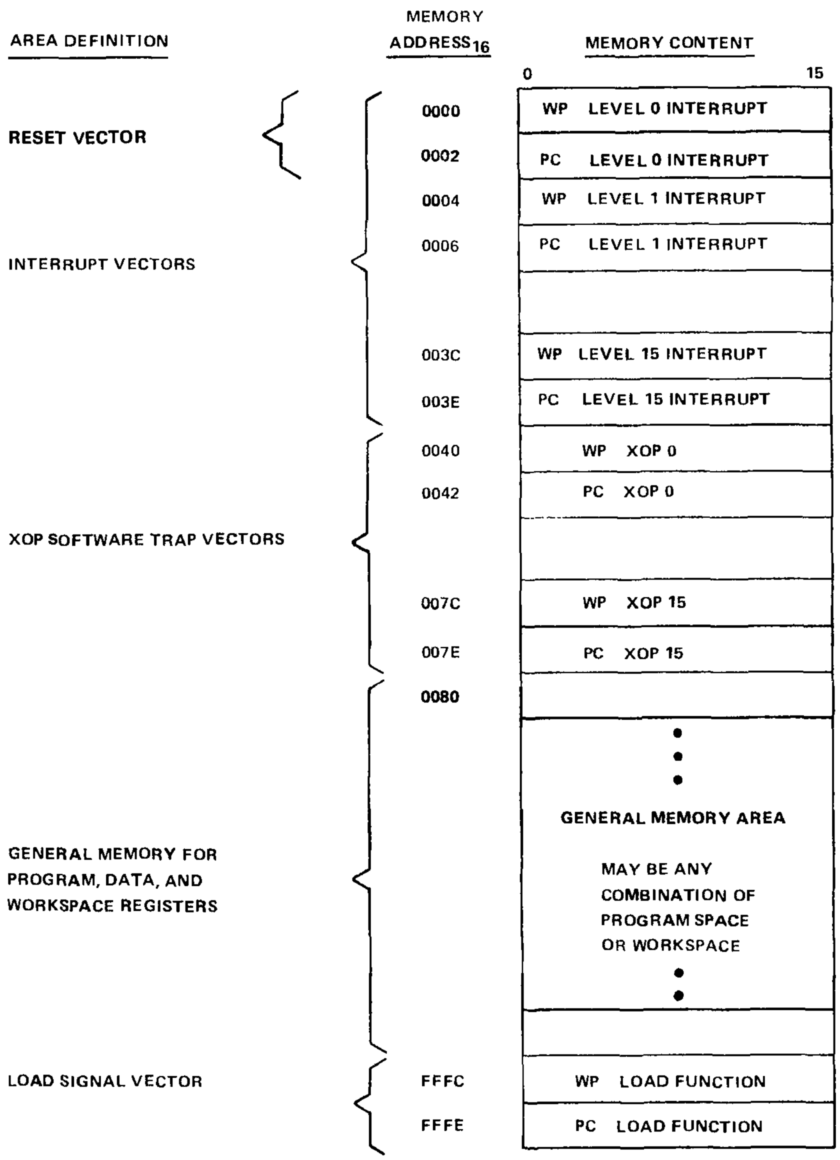


Figure 4-12. *TMS 9900 Dedicated Memory Addresses*

RESET Vector

The first two memory words are reserved for storage of the RESET vector. The RESET vector is used to load the new WP and PC whenever the CPU RESET signal occurs. The first word contains the new WP, which is the starting address of the RESET workspace. The second word contains the new PC, which is the starting address of the RESET service routine.

Interrupt Vectors

The next thirty memory words, 0004_{16} through $003E_{16}$ are reserved for storage of the interrupt transfer vectors for levels 1 through 15. Each interrupt level uses a word for the workspace pointer (WP) and a word for the starting address of the service routine (PC). If an interrupt level is not used within a system, then the corresponding two memory words can be used for program or data storage.

Software Trap Vectors

The next thirty-two memory words, 0040_{16} through $007E_{16}$, are used for extended-operation software trap vectors. When the CPU executes one of the 16 extended operations (XOPs), the program traps through the corresponding vector. Two words are reserved for each trap vector, with one word for the WP and one word for the PC. If an XOP instruction is not used, the corresponding vector words can be used for program or data storage.

LOAD Vector

The last two memory words $FFFC_{16}$ and $FFFE_{16}$ are reserved for the LOAD vector, with one word for the WP and one word for the PC. The LOAD vector is used whenever the CPU LOAD signal is active (low).

Transfer Vector Storage

The transfer vectors can be stored either in ROM or RAM, but either the RESET or LOAD vector should be in non-volatile memory and should point to a program in non-volatile storage to ensure proper system start-up. The restart routine should initialize any vector which is in RAM. The program can then manipulate the RAM-based vectors to alter workspace assignments or service routine entry points, while ROM-based vectors are fixed and cannot be altered.

Memory Control Signals

The 9900 uses three signals to control the use of the data bus and address bus during memory read or write cycles. Memory enable ($\overline{\text{MEMEN}}$) is active low during all memory cycles.

Data bus in (DBIN) is active high during memory read cycles and indicates that the CPU has disabled the output data buffers.

Write enable ($\overline{\text{WE}}$) is active low during memory write cycles and has timing compatible with the read/write (R/$\overline{\text{W}}$) control signal for many standard RAMs.

Memory Read Cycle

Figure 4-13 illustrates the timing for a memory read machine cycle with no wait states. At the beginning of the machine cycle, $\overline{\text{MEMEN}}$ and DBIN become active and the valid address is output on A0-A14. D0-D15 output drivers are disabled to avoid conflicts with input data. $\overline{\text{WE}}$ remains high for the entire machine cycle. The READY input is sampled on Φ1 of clock cycle 1, and must be high if no wait states are desired. Data is sampled on Φ1 of clock cycle 2, and set-up and hold timing requirements must be observed. A memory-read cycle is never followed by a memory-write cycle, and D0-D15 output drivers remain disabled for at least one additional clock cycle.

Memory Write Cycle

Figure 4-14 illustrates the timing for a memory write machine cycle with no wait states. $\overline{\text{MEMEN}}$ becomes active, and valid address and data are output at the beginning of the machine cycle. DBIN remains inactive for the complete cycle. $\overline{\text{WE}}$ goes low on Φ1 of clock cycle 1 and goes high on Φ1 of clock cycle 2, meeting the address and data set-up and hold timing requirements for the static RAMs listed in Chapter 2. For no wait states, READY must be high during Φ1 of clock cycle 1.

Read/Write Control with DBIN

In some memory systems, particularly with dynamic RAMs, it may be desirable to have READ and WRITE control signals active during the full memory cycle. *Figure 4-15* shows how the WRITE signal can be generated. Note that DBIN is high *only* for READ cycles; therefore, $\overline{\text{MEMEN}}$ can be NORed with DBIN to yield a WRITE signal which is high only during memory write operations.

Slow Memory Control

Although most memories operate with the 9900 at the full system speed, some memories cannot properly respond within the minimum access time determined by the system clock. The system clock could be slowed down in order to lengthen the access time but the system through-put would be adversely affected since non-memory and other memory reference cycles would be unnecessarily longer. The READY and WAIT signals are used instead to synchronize the CPU with slow memories. The timing for memory-read and memory-write cycles with wait states is shown in *Figures 4-16* and *4-17*.

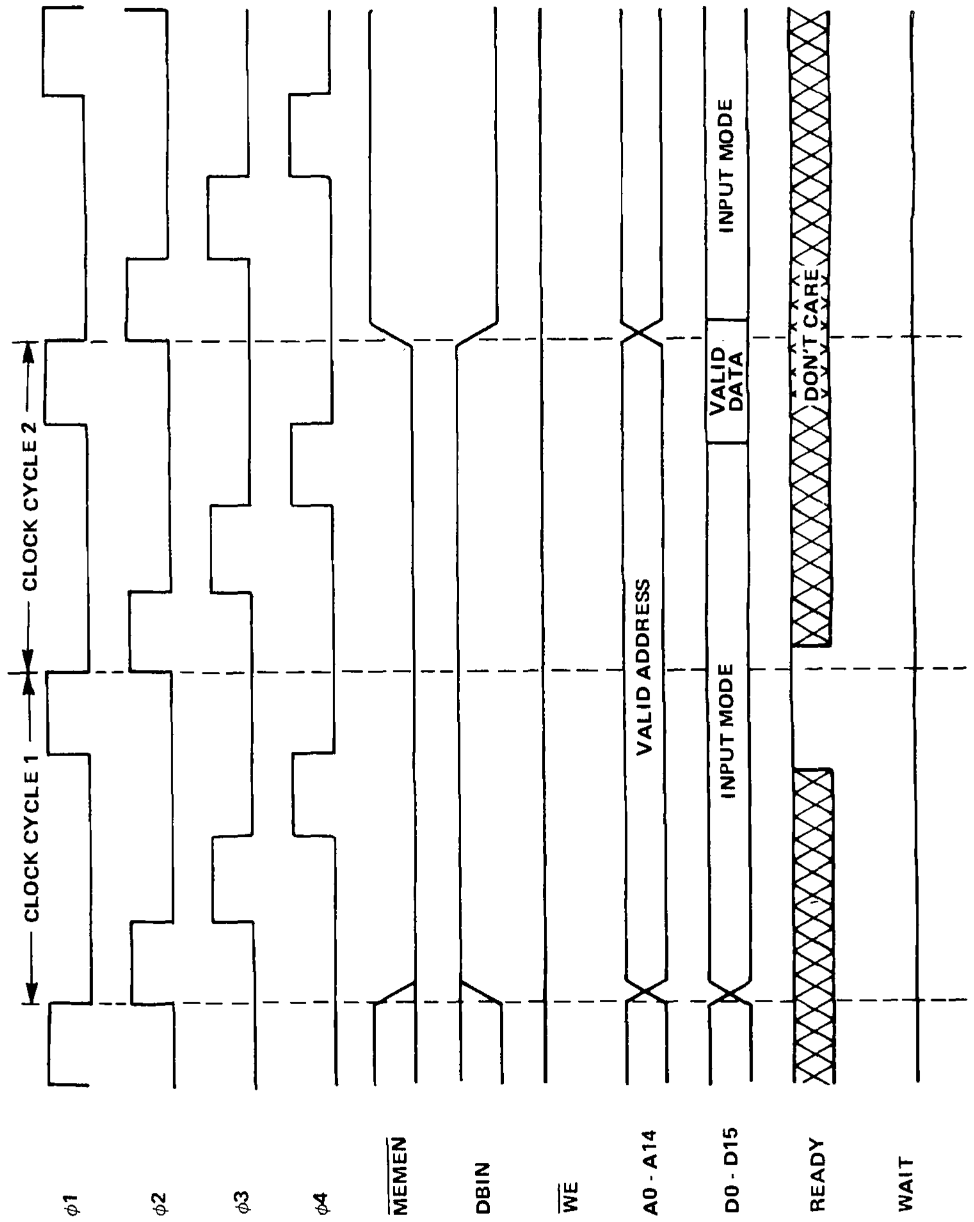


Figure 4-13. Memory-Read Cycle Timing

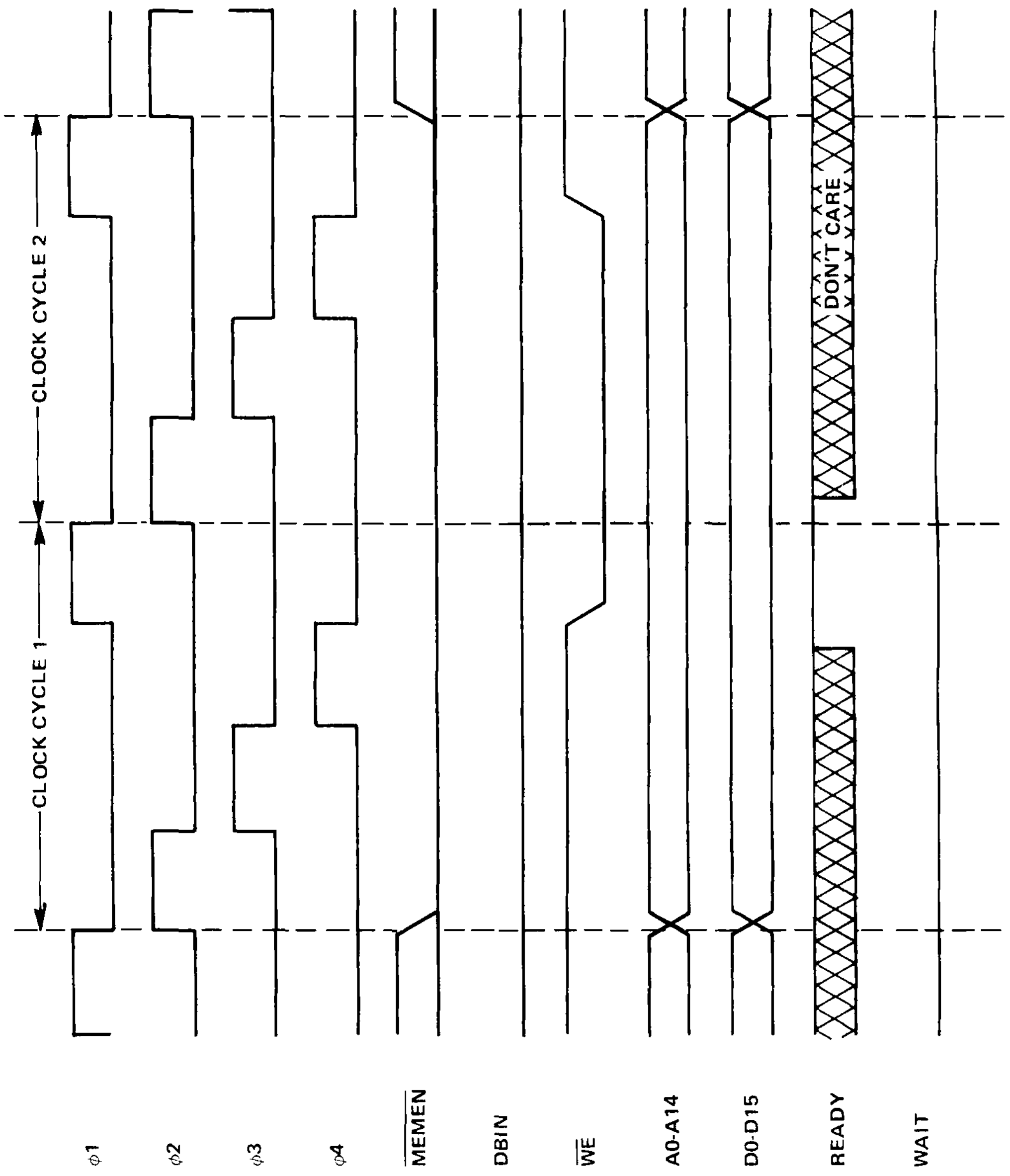


Figure 4-14. Memory-Write Cycle Timing

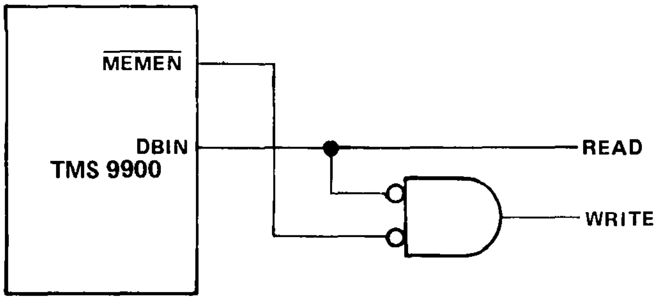


Figure 4-15. Read/Write Control Using $\overline{MEMEN}$ and DBIN

The READY input is tested on Φ1 of clock cycle 1 of memory-read and memory-write cycles. If READY = 1, no wait states are used and the data transfer is completed on the next clock cycle. If READY = 0, the processor enters the wait state on the next clock cycle and all memory control, address, and data signals maintain their current levels. The WAIT output goes high on Φ3 to indicate that a wait state has been entered. While in the wait state, the processor continues to sample READY on Φ1, and remains in the wait state until READY = 1. When READY = 1 the processor progresses to clock cycle 2 and the data transfer is completed. WAIT goes low on Φ3. It is important to note that READY is only tested during Φ1, of clock cycle 1 of memory-read and memory-write cycles and wait states, and the specified set-up and hold timing requirements must be met; at any other time the READY input may assume any value. The effect of inserting wait states into memory access cycles is to extend the minimum allowable access time by one clock period for each wait state.

▶4

Wait State Control

Figure 4-18 illustrates the connection of the WAIT output to the READY input to generate one wait state for a selected memory segment. The address decode circuity generates an active low signal ($\overline{\text{SLOMEM}}$ = 0) whenever the slow memory is addressed. For example, if memory addresses 8000_{16} – $FFFE_{16}$ select slow memory, $\overline{\text{SLOMEM}}$ = A0. If one wait state is required for all memory, WAIT may be connected directly to READY, causing one wait state to be generated on each memory-read or memory-write machine cycle. Referring again to *Figures 4-16* and *4-17* note that the WAIT output satisfies all of the timing requirements for the READY input for a single wait state. The address decode signal is active only when a particular set of memory locations has been addressed. *Figure 4-19* illustrates the generation of two wait states for selected memory by simply delaying propagation of the WAIT output to the READY input one clock cycle with a D-type flip-flop. The rising edge of Φ2TTL is assumed to be coincident with the falling edge of the Φ2 clock input to the TMS 9900.

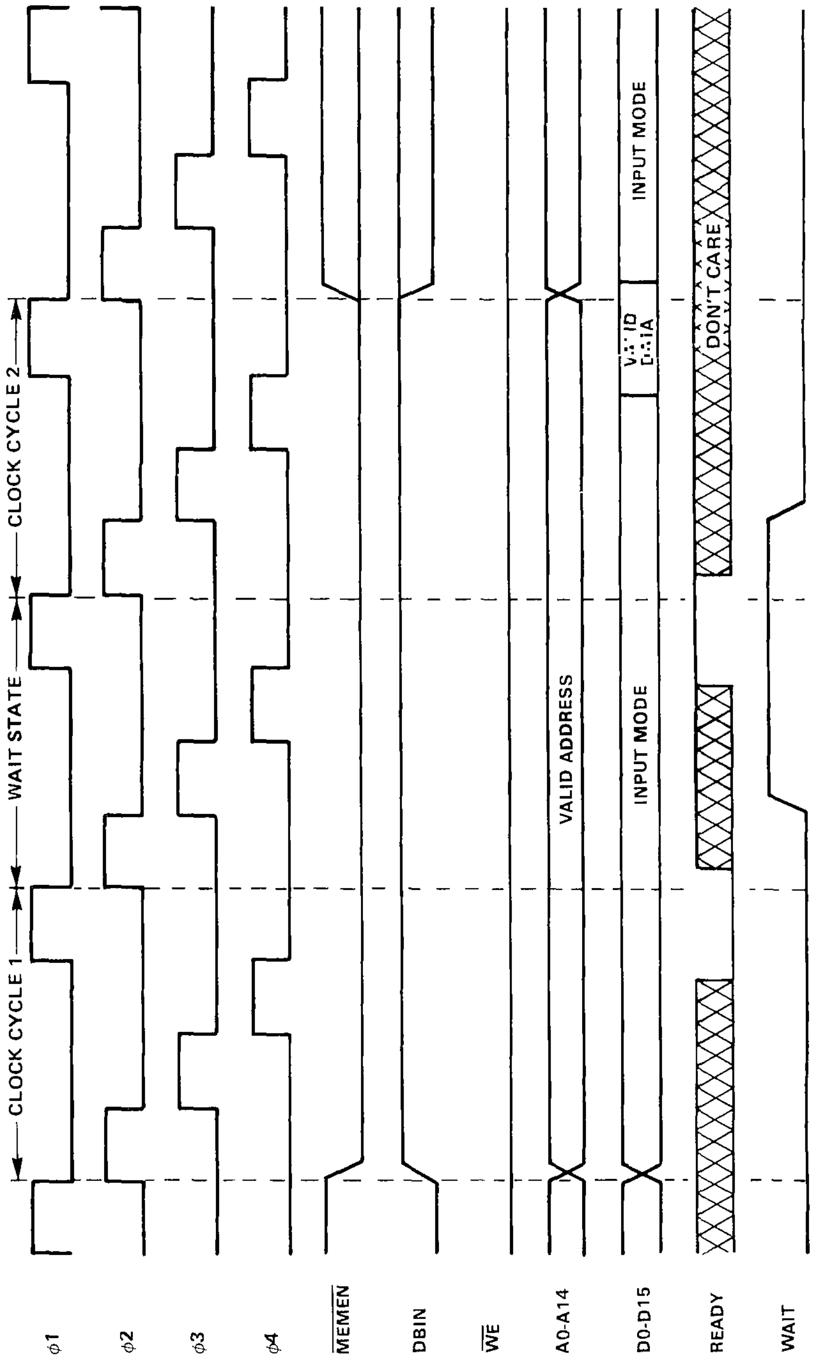


Figure 4-16. Memory-Read Cycle With One Wait State

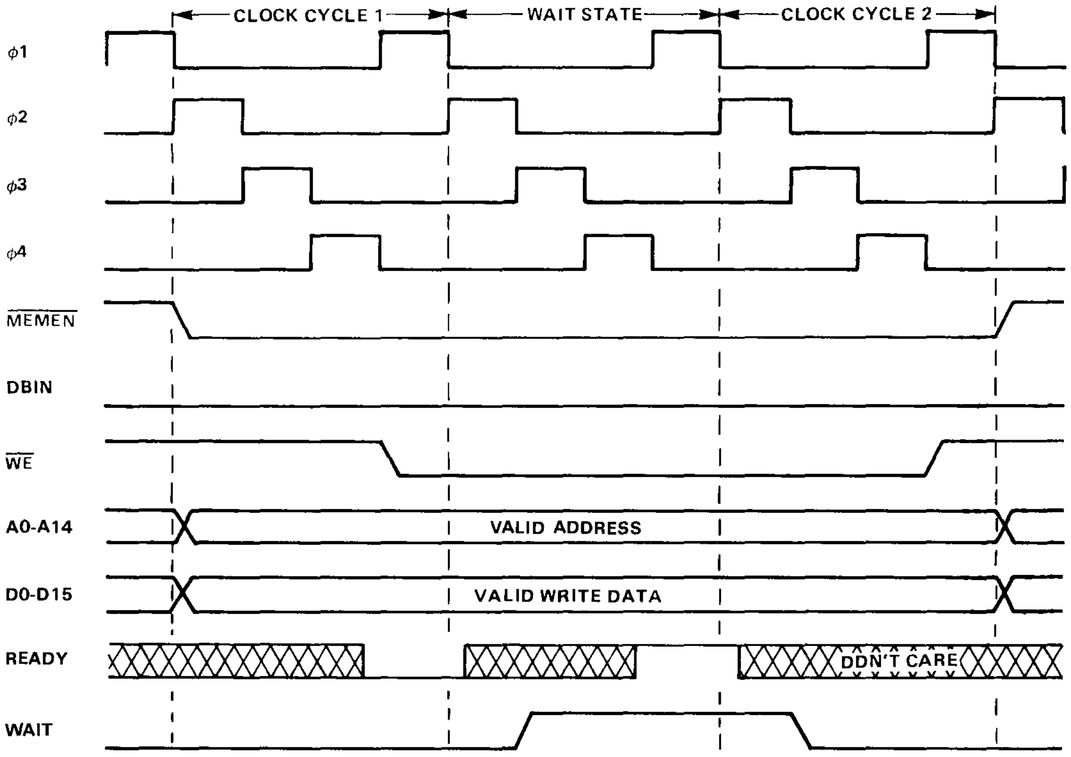


Figure 4-17. Memory-Write Cycle With One Wait State

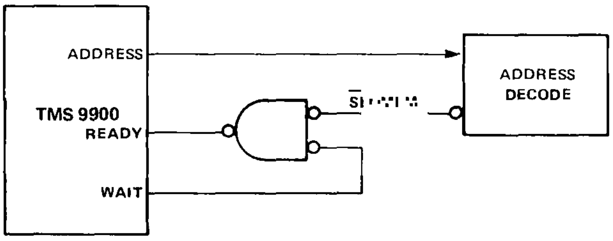


Figure 4-18. Single Wait State for Slow Memory

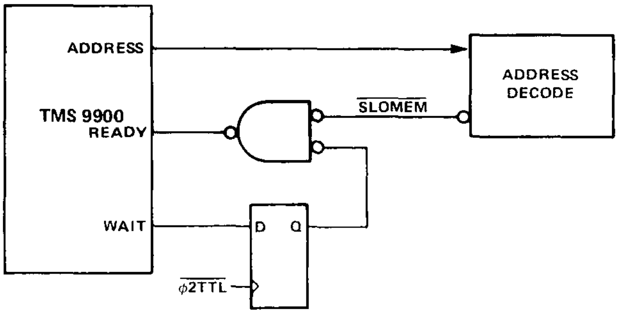


Figure 4-19. Double Wait States for Slow Memory

Memory Access Time Calculation

Maximum allowable memory access time for the TMS 9900 can be determined with the aid of *Figure 4-20.* Memory control and address signals are output on $\Phi 2$ of clock cycle 1, and are stable 20 ns (t_{PLH}, t_{PHL}) afterwards. Data from memory must be valid 40 ns (t_{su}) before the leading edge of $\Phi 1$ during clock cycle 2. Therefore, memory access time may be expressed by the equation:

$$t_{acc} \leq (1.75 + n)\, t_{cy} - t_{PLH} - t_r - t_{su}$$

where n equals the number of wait states in the memory-read cycle. Assigning worst-case specified values for t_{PLH} (20ns), t_r (12ns), and t_{su} (40 ns), and assuming 3 MHz operation:

$$t_{acc} \leq \frac{(1.75 + n)}{0.003} - 72 \text{ ns}$$

Access time is further reduced by address decoding, control signal gating, and address and data bus buffering, when used. Thus, for a known access time for a given device, the number of required wait states can be determined.

For example, a TMS 4042-2 RAM has a 450 nanosecond access time and does not require any wait states. A TMS 4042 has a 1000 nanosecond access time and requires two wait states. Propagation delays caused by address or data buffers should be added to the nominal device access time in order to determine the effective access time.

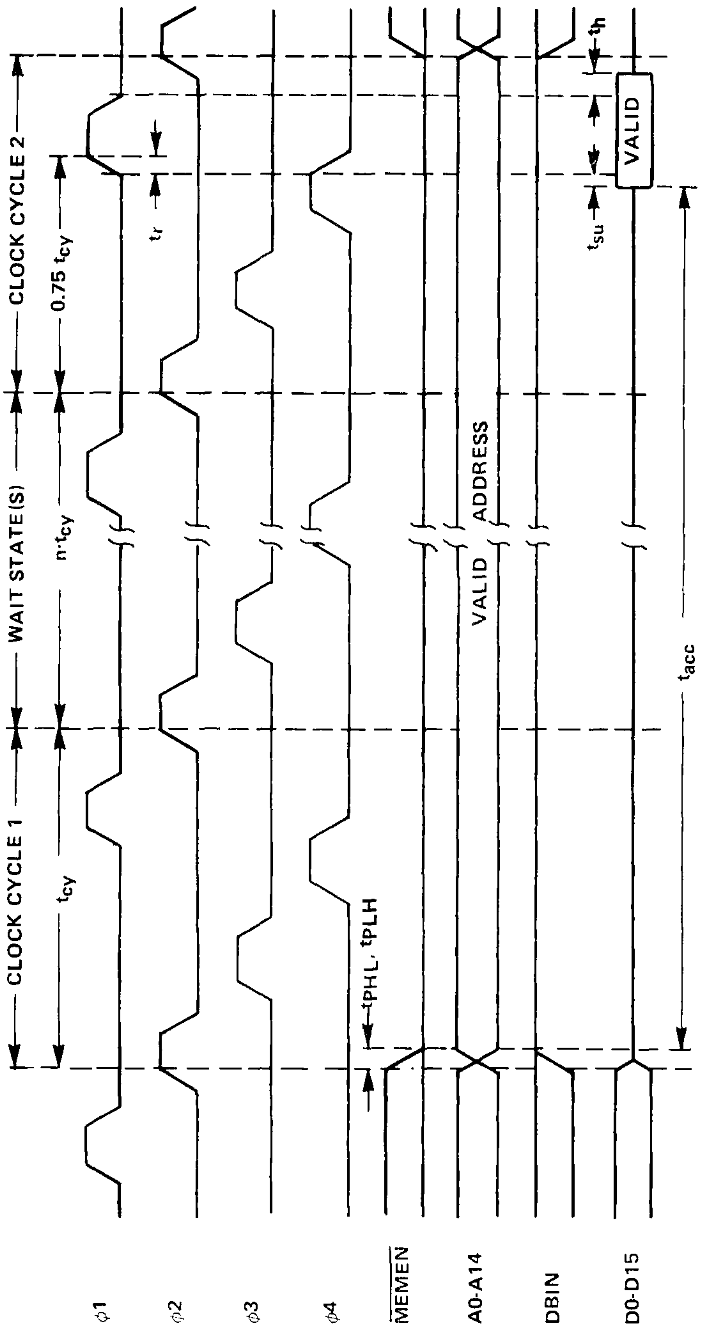


Figure 4-20. Memory Access Timing Calculation

Static Memory

Static RAMs and PROMs are easily interfaced to the 9900. A 9900 memory system using the TMS 4042-2 256 X 4 static RAM and the TMS 2708 1K X 8 EPROM is shown in *Figure 4-21.*

Address

The most-significant address bit, A0, is used to select either the EPROMs or the RAMs during memory cycles. When A0 is low, the EPROMs are selected, and when A0 is high, the RAMs are selected. Address lines A1 through A4 are not used since the full address space of the TMS 9900 is not required in the example. The lower address bits select internal RAM or EPROM cells. Other memory systems can fully decode the address word for maximum memory expansion.

Control Signals

Since DBIN is also used to select the EPROMs during memory-write cycles, the EPROMs cannot inadvertently be selected and placed into output mode while the CPU is also in the output mode on the data bus. $\overline{\text{MEMEN}}$ is used to select the RAMs during either read or write cycles, and $\overline{\text{WE}}$ is used to select the read/write mode. DBIN is also used to control the RAM output bus drivers.

The 9900 outputs $\overline{\text{WE}}$ three clock phases after the address, data, and $\overline{\text{MEMEN}}$ are output. As a result, the address, data, and enable-hold times are easily met. $\overline{\text{WE}}$ is enabled for one clock cycle and satisfies the minimum write pulse width requirement of 300 nanoseconds. Finally, $\overline{\text{WE}}$ is disabled one clock phase before the address, data, and other control signals and meets the TMS 4042-2 50-nanosecond minimum data and address hold time.

Loading

The loads on the CPU and memory outputs are well below the maximum rated loads. As a result no buffering is required for the memory system in *Figure 4-21.* The TMS 4042-2 and the TMS 2708 access times are low enough to eliminate the need for wait states, and the CPU READY input is connected to V_{CC}.

The minimum high-level input voltage of the TMS 2708 is 3 volts while the maximum high-output voltage for the TMS 9900 is 2.4 volts at the maximum specified loading. For the system in *Figure 4-21,* the loads on the CPU and memory outputs are well below the maximum rated load. At this loading, the TMS9900 output voltage exceeds 3 volts, so pull-up resisters are not needed.

There are many other Texas Instruments static memories compatible with the TMS 9900. Most memory devices do not require wait states when used with the TMS 9900 at 3 MHz.

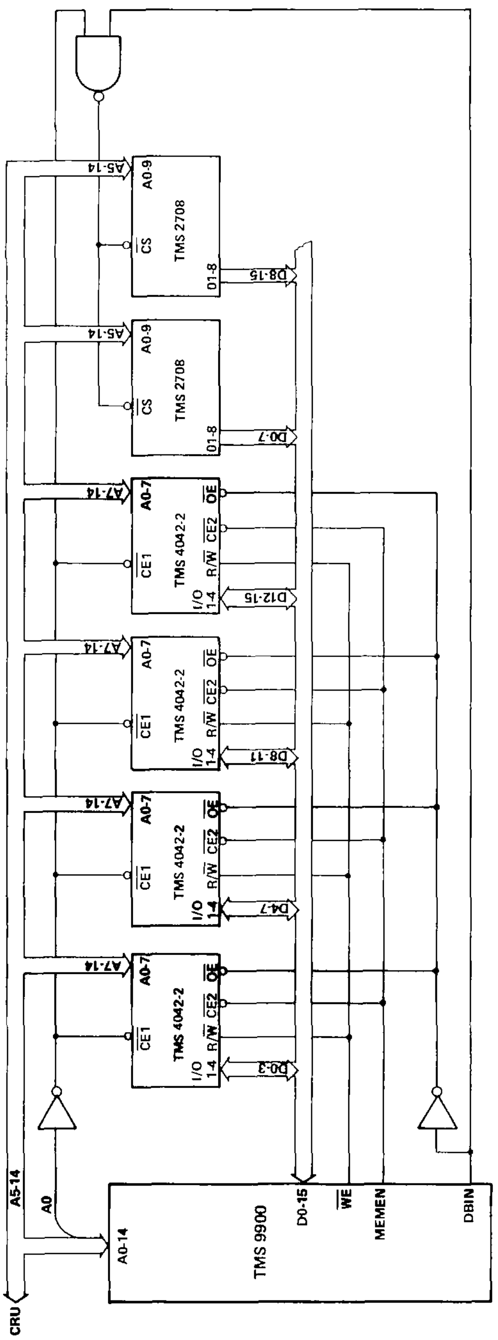


Figure 4-21. TMS 9900 Static Memory System

Dynamic Memory

Memory applications requiring large bit storage can use 4K, 16K or 64K dynamic memories for low cost, low power consumption, and high bit density. TMS 9900 systems requiring 4K words or more of RAM, can economically use the 4096-bit TMS 4051, the 16,384-bit TMS 4116, or any of the other dynamic RAMs covered in Chapter 2.

Refresh

The dynamic RAMs must be refreshed periodically to avoid the loss of stored data. The RAM data cells are organized into a matrix of rows and columns with on-chip gating to select the addressed bit. Refresh of the 4K RAM cell matrix is accomplished by performing a memory cycle of each of the 64 row addresses every 2 milliseconds or less. The 16K RAM has 128 row addresses. Performing a memory cycle at any cell on a row refreshes all cells in the row, thus allowing the use of arbitrary column address during refresh.

Refresh Modes

There are several dynamic memory refresh techniques which can be used for a TMS 9900 system. If the system periodically accesses at least one cell of each row every 2 milliseconds, then no additional refresh circuitry is required. A CRT controller which periodically refreshes the display, illustrates this concept.

Refresh control logic is included wherever the system cannot otherwise ensure that all rows are refreshed every 2 milliseconds. The dynamic memory in such TMS 9900 systems can be refreshed in the block, cycle stealing, or transparent mode.

Block Refresh.

The block mode of refresh halts the CPU every 2 milliseconds and sequentially refreshes each of the rows. The block technique halts execution for a 128 (4K) or 256 (16K) clock cycle periods every 2 milliseconds. Some TMS 9900 systems cannot use this technique because of the possibility of slow response to priority interrupts or because of the effect of the delay during critical timing or I/O routines.

Cycle Stealing.

The cycle stealing mode of refresh "steals" a cycle from the system periodically to refresh one row. The refresh interval is determined by the maximum refresh time and the number of rows to be refreshed. The 4K dynamic RAMs have 64 rows to be refreshed every 2 milliseconds and thus require a maximum cycle stealing interval of 31.2 microseconds.

A cycle stealing refresh controller for the TMS 4051 4K dynamic RAM is shown in *Figure 4-22.* The refresh timer generates the refresh signal (RFPLS) every 30 microseconds. The refresh request signal (RFREQ) is true until the refresh cycle is completed. The refresh grant signal (RFGNT) goes high during the next CPU clock cycle in which the CPU is not accessing the dynamic memory. The refresh memory cycle takes two clock cycles to complete after RFGNT is true. During the second clock cycle, however, the CPU can attempt to access the dynamic memory since the CPU is not synchronized to the refresh controller. If the CPU does access memory during the last clock cycle of the refresh memory cycle, the refresh controller makes the memory not-ready for the remainder of the refresh memory cycle, and the CPU enters a wait state during this interval. The dynamic memory row address during the refresh memory cycle is the output of a modulo-64 counter. The counter is incremented each refresh cycle in order to refresh the rows sequentially.

The dynamic memory timing controller generates the proper chip enable timing for both CPU and refresh initiated memory cycles. The timing controller can be easily modified to operate with other dynamic RAMs.

Since the TMS 9900 performs no more than three consecutive memory cycles, the refresh request will be granted in a maximum of three memory cycles. Some systems may have block DMA, which uses $\overline{\text{HOLD}}$. RFREQ can be used in such systems to disable HOLDA temporarily in order to perform a refresh memory cycle if the DMA block transfer is relatively long (greater than 30 microseconds). The cycle stealing mode "steals" clock cycles only when the CPU attempts to access the dynamic memory during the last half of the refresh cycle. Even if this interference occurs during each refresh cycle, a maximum of 64 clock cycles are "stolen" for refresh every 2 milliseconds.

Transparent Refresh.

The transparent refresh mode eliminates this interference by synchronizing the refresh cycle to the CPU memory cycle. The rising edge of $\overline{\text{MEMEN}}$ marks the end of a memory cycle immediately preceding a non-memory cycle. The $\overline{\text{MEMEN}}$ rising edge can initiate a refresh cycle with no interference with memory cycles. The refresh requirement does not interfere with the system throughput since only non-memory cycles are used for the refresh cycles. The worst-case TMS 9900 instruction execution sequence (all divides) will guarantee the complete refresh of a 4K or 16K dynamic RAM within 2 milliseconds.

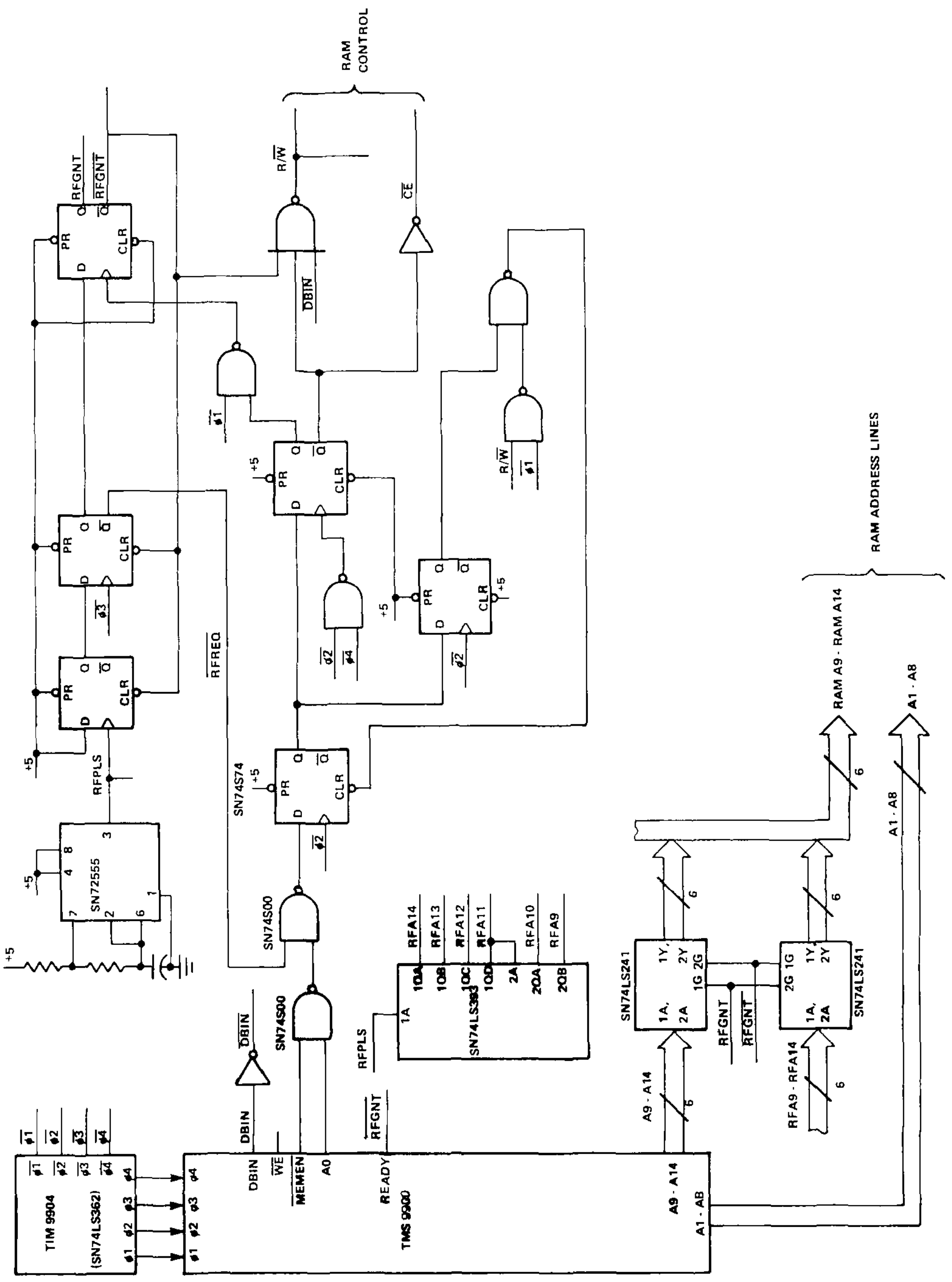


Figure 4-22. Cycle-Stealing Dynamic RAM Refresh for TMS 4051

While the transparent refresh mode eliminates refresh-related system performance degradation, the system power consumption can be higher since the RAMs are refreshed more often than required. As many as one-half of the CPU machine cycles can be refresh cycles, resulting in multiple refresh cycles for each row during the refresh interval. This situation can be corrected by adding a timer to determine the start of the refresh interval and an overflow detector for the refresh row counter. When every row has been refreshed during an interval, the refresh circuit is disabled until the beginning of the next interval. Since each row is refreshed only once, the system power consumption is reduced to a minimum.

Direct memory access using $\overline{\text{HOLD}}$ should guarantee that sufficient non-memory cycles are available for refresh during large block transfers. An additional refresh timer can be used to block HOLDA in order to provide periodic refresh cycles.

BUFFERED MEMORY

▶4 The TMS 9900 outputs can drive approximately two standard TTL inputs and 200 picofarads. Higher capacitive loads may be driven, but with increased rise and fall times. Many small memory systems can thus be directly connected to the CPU without buffer circuits. Larger memory systems, however, may require external bipolar buffers to drive the address or data buses because of increased loading. Texas Instruments manufactures a number of buffer circuits compatible with the TMS 9900. The SN74LS241 noninverting-octal buffer with three-state outputs is an example of a buffer circuit.

A TMS 9900 memory system with address and data bus buffering is shown in *Figure 4-23.* The system consists of sets of four 256 X 4 memory devices in parallel to provide the 16-bit data word. The four sets of four devices provide a total of 1024 words of memory. The memory devices can be the TMS 4042-2 NMOS static RAM.

The SN74S412 octal buffer/latch is designed to provide a minimum high-level output voltage of 3.65 V. Buffered TMS 9900 memory systems containing the TMS 4700 ROM or the TMS 2708 EPROM, for example, require input voltages in excess of the output voltages of many buffer circuits. The SN74S412 can be used to buffer the memories without the pull-up resistors needed for buffers.

MEMORY PARITY

Parity or other error detection/correction schemes are often used to minimize the effects of memory errors. Error detection schemes such as parity are used to indicate the presence of bad data, while error correction schemes correct single or multiple errors.

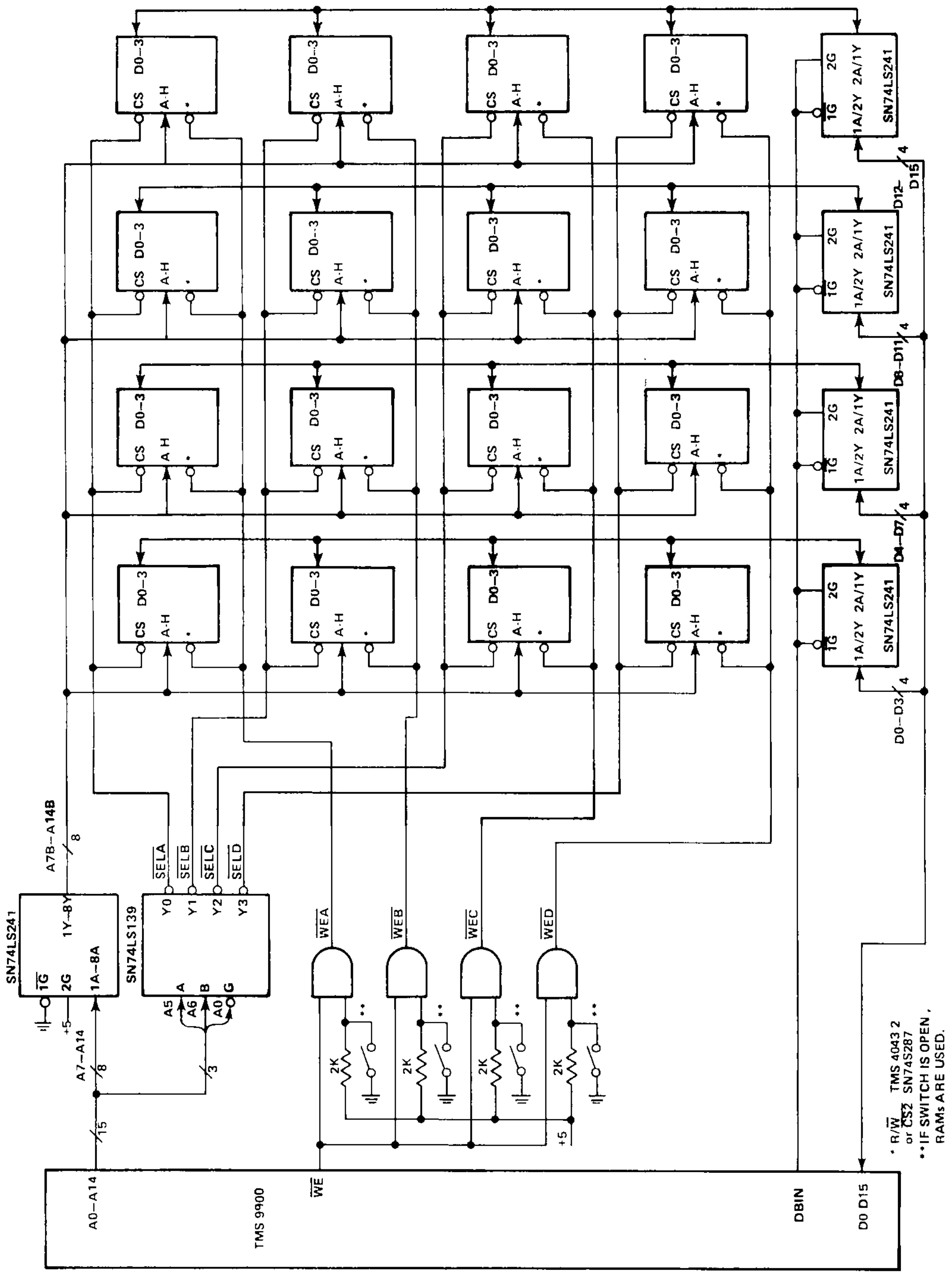


Figure 4-23. Buffered Memory with Mixed PROM/ROM

The SN74LS280 parity generator/checker can be used to implement memory parity in a TMS 9900 system. The system in *Figure 4-24* uses two SN74LS280 circuits to generate and to check the odd-memory parity. During memory write cycles, the generated parity bit is output to bit D16 of the memory. During memory read cycles, the parity is checked and an interrupt, $\overline{\text{PARERR}}$, is generated if the parity is even.

It should be noted that the faulty memory word will have already been used by the CPU as an op code, address, or data before the interrupt is generated. This can cause trouble in determining the exact location of the error. For example, an error in bit 8 of the CLR op code will cause the CPU to branch unconditionally. When the interrupt is serviced, there would then be no linkage to the part of the program at which the error occurred. A diagnostic routine can often isolate such errors by scanning the memory and checking parity under program control. Such a parity error in the diagnostic itself can be extremely difficult to isolate.

An external address latch clocked at IAQ can be used to retain program linkage under the above circumstances. When the parity error is detected, the address latch is frozen, thus pointing to the address of the instruction during which the parity error occurred.

Memory Layout

It is generally advantageous to lay out memory devices as arrays in the system. The advantages are twofold. First, positioning the devices in an orderly fashion simplifies identification of a particular memory element when troubleshooting. Second, and most important, layout of memory arrays simplifies layout, shortens interconnections, and generally allows a more compact and efficient utilization of board space. Crosstalk between adjacent lines in memory arrays is minimized by running address and data lines parallel to each other, and by running chip enable signals perpendicular to the address lines.

Memory devices, particularly dynamic RAMs generally require substantially greater supply currents when addressed than otherwise. It is therefore important that all power and ground paths be as wide as possible to memory arrays. Furthermore, in order to avoid spikes in supply voltages, it is advisable to decouple supply voltages with capacitors as close as possible to the pins of the memory devices. As an example, a system containing a 4K x 16-bit array of TMS 4051s should contain one 15 μF and one 0.05 μF capacitor for each set of four memory devices; with the large capacitors decoupling V_{DD}, and the small capacitors decoupling V_{BB}.

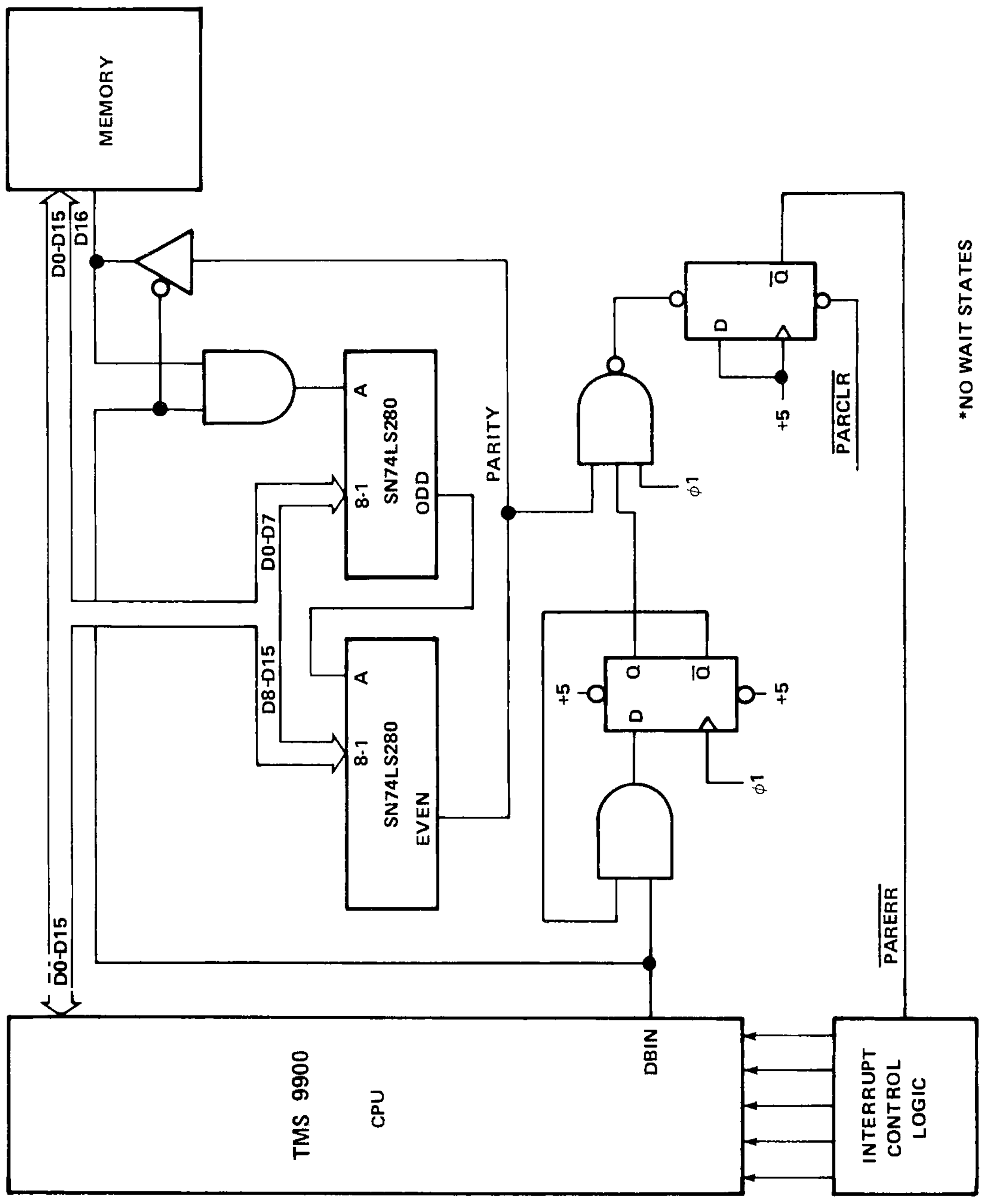


Figure 4-24. Memory Parity Generator Checker

INSTRUCTION EXECUTION

Execution time for an instruction is a function of the clock frequency, the number of clock cycles, the number of memory accesses and the number of wait states if required for slower memories. The following tables list the number of clock cycles required to execute each instruction if no wait states are required. The number of memory accesses is also given so that the extra clock cycles can be calculated for the number of wait states required. A wait state is entered when the ready signal from the memory does not go high within one clock period after initiation of a memory cycle. For example: The clock frequency for the TMS 9900 is 3 MHz. From the calculation of maximum access time for no wait states, the memory access time must be less than 512 ns. One wait state (of 333 ns duration) will be required for memories with access times between 512 ns and 845 ns, two wait states will be required if the access time is between 845 ns and 1.178 μ sec, and so on.

▶4

TIMING

From *Figure 4-25,* the first execution time table, an add instruction (A) using direct register addressing for both operands requires 14 clock cycles if there are no wait states required. For other addressing modes, the number of clock cycles increases to a maximum of 30. If the memory requires one wait state per access, an additional four clock periods will be required since there are four memory cycles in the execution of an add instruction. For the TMS 9900 running at 3 MHz, 14 clock periods will take 4.667 microseconds; 30 clock periods will take 10.0 microseconds. The number of memory cycles is from 4 up to 8 depending upon addressing mode (3 to 7 for compare, C). Use the tables in the following manner. Assuming one wait state, a clock frequency of 3 MHz, and an instruction with complex addressing, the tables can be used to determine the execution time for the instruction

```
A *R1, @ LIST
```

is 26 clock cycles for fetch and execution and 6 clock cycles for wait states, or 32 x .333 microseconds which is 10.667 microseconds.

Figures 4-26, 27 and *28* give the rest of the execution time data, always by number of clock cycles (assuming no wait states) and memory cycles. Execution times for instructions which do not use the five general addressing modes may be found in Chapter 8 in the *CPU* section.

INSTRUCTIONS A, C†, S, SOC, SZC, MOV

	Destination Address	Source Address				
		R	*R	*R +	@LIST	@TABLE (R)
Clock Cycles	R	14	18	22	22	22
	*R	18	22	26	26	26
	*R +	22	26	30	30	30
	@LIST	22	26	30	30	30
	@TABLE (R)	22	26	30	30	30
Memory Cycles	R	4	5	6	5	6
	*R	5	6	7	6	7
	*R +	6	7	8	7	8
	@LIST	5	6	7	6	7
	@TABLE (R)	6	7	8	7	8
†Memory Cycles for C instr.	R	3	4	5	4	5
	*R	4	5	6	5	6
	*R +	5	6	7	6	7
	@LIST	4	5	6	5	6
	@TABLE (R)	5	5	6	7	6

Figure 4-25.

INSTRUCTIONS: AB, CB††, SB. SOCB, SZCB, MOVB

	Destination Address	Source Address				
		R	*R	*R +	@LIST	@TABLE (R)
Clock Cycles	R	14	18	20	22	22
	*R	18	22	24	26	26
	*R +	20	24	26	28	28
	@LIST	22	26	28	28	28
	@TABLE (R)	22	26	28	28	28
Memory Cycles	R	4	5	6	5	6
	*R	5	6	7	6	7
	*R +	6	7	8	7	8
	@LIST	5	6	7	6	7
	@TABLE (R)	6	7	8	7	8
††Memory Cycles for CB instr.	R	3	4	5	4	5
	*R	4	5	6	5	6
	*R +	5	6	7	6	7
	@LIST	4	5	6	5	6
	@TABLE (R)	5	6	7	6	7

Figure 4-26.

INSTRUCTIONS LDCR, STCR

LDCR

	Addressing Mode	Bit Count, C															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Clock Cycles	R	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52
	*R	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56
	*R +	28	30	32	34	36	38	40	42	46	48	50	52	54	56	58	60
	@LIST	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60
	@TABLE (R)	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60
Memory Cycles	R	3								3							
	*R	4								4							
	*R +	5								5							
	@LIST	4								4							
	@TABLE (R)	5								5							

STCR

	Addressing Mode	Bit Count, C															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
Clock Cycles	R	42	42	42	42	42	42	42	44	58	58	58	58	58	58	58	60
	*R	46	46	46	46	46	46	46	48	62	62	62	62	62	62	62	64
	*R +	48	48	48	48	48	48	48	50	66	66	66	66	66	66	66	68
	@LIST	50	50	50	50	50	50	50	52	66	66	66	66	66	66	66	68
	@TABLE (R)	50	50	50	50	50	50	50	52	66	66	66	66	66	66	66	68
Memory Cycles	R	4								4							
	*R	5								5							
	*R..	6								6							
	@LIST	5								5							
	@TABLE (R)	6								6							

Figure 4-27.

	Clock Cycles					Memory Cycles				
Instruction	R	*R	*R+	@LIST,	@TABLE (R)	R	*R	*R+	@LIST	@TABLE (R)
ABS MSB = 0	12	16	20	20	20	2	3	4	3	4
MSB = 1	14	18	22	22	22	3	4	5	4	5
B	8	12	16	16	16	2	3	4	3	4
BL	12	16	20	20	20	3	4	5	4	5
BLWP	26	30	34	34	34	6	7	8	7	8
CLR	10	14	18	18	18	3	4	5	4	5
DEC	10	14	18	18	18	3	4	5	4	5
DECT	10	14	18	18	18	3	4	5	4	5
INC	10	14	18	18	18	3	4	5	4	5
INCT	10	14	18	18	18	3	4	5	4	5
INV	10	14	18	18	18	3	4	5	4	5
NEG	12	16	20	20	20	3	4	5	4	5
SETO	10	14	18	18	18	3	4	5	4	5
SWPB	10	14	18	18	18	3	4	5	4	5
XOP	36	40	44	44	44	8	9	8	9	8
XOR	14	18	22	22	22	4	5	6	5	6

Figure 4-28.

CYCLIC OPERATION

An example of a machine cycle sequence is illustrated in *Figure 4-29.* For an add instruction the machine cycles alternate between memory cycles and ALU cycles. The first cycle is always a memory read cycle to fetch the instruction and the second is always an ALU cycle to decode the instruction. Each machine cycle requires two clock cycles, thus the 7 machine cycles shown for the add instruction require 14 clock cycles.

A R1, R2

1	Memory Read	Instruction Fetch
2	ALU	Decode Opcode
3	Memory Read	Fetch (WR1)
4	ALU	Set Up
5	Memory Read	Fetch (WR2)
6	ALU	Addition
7	Memory Write	Store Result in WR2 and Increment PC

Figure 4-29. Machine Cycles for an Add Instruction

The 9900 performs its functions under control of a 4-phase clock and, fundamentally, performs instruction fetch and execution cycles. *Figure 4-30* illustrates the step-by-step procedure the 9900 uses to execute an add instruction. From previous cycles, the workspace pointer has been loaded with the number 0800, and the program counter contains the number 0100.

Step 1. The first step in any instruction cycle is to fetch the instruction. This is accomplished by gating the content of the program counter into the memory address register. The output of the memory address register is the address bus which is connected to the memory. In this case, word number 0100 is read from the memory and placed in the instruction register on the 9900 chip. From this point, the ones and zeros of the instruction register control the sequence of microcode stored in the microcontrol read only memory on the 9900 chip. These microsteps become the execution phase of the instruction.

Step 2. At this point, the microcontrol shifts to the execution of an add instruction; the first operand must be obtained from memory. In order to do this, the workspace pointer and a portion of the instruction word (the source operand register number) are added together via the ALU and placed in the memory address register.

Step 3. The address 0802 is the result (in this example), and being supplied to the memory produces on the data bus the content of memory word 0802 which is the binary equivalent of 25. This number must be stored in a temporary register on the 9900 chip, in this case the T1 register.

Step 4. Now a second operand must be fetched. Again the workspace pointer is added to the content of that portion of the instruction word which is the destination register identifier. The sum of these two is 0804 for register two, and this number is placed in memory address register and goes out on the address bus.

Step 5. Memory word 0804 is read and the number 10 is brought into the 9900 chip. The register which stores the second operand is called the source data register or S register.

Step 6. At this point the two operands have been loaded into registers on the 9900 chip and may be added by the ALU to produce the result. Register T1 containing 25 is added to the register S which contains 10 and the sum, 35, replaces the 10 in the S register and is placed on the data bus via the S register.

Step 7. The address bus still contains the number 0804 which was the address of the second operand and is the location in memory where the result is to be stored. So at this point in the cycle, a memory write cycle is initiated and the binary equivalent of 35 is stored in memory location 0804. At the conclusion of this memory cycle the program counter is incremented by two to point to the next sequential memory word, which is the instruction to be executed next.

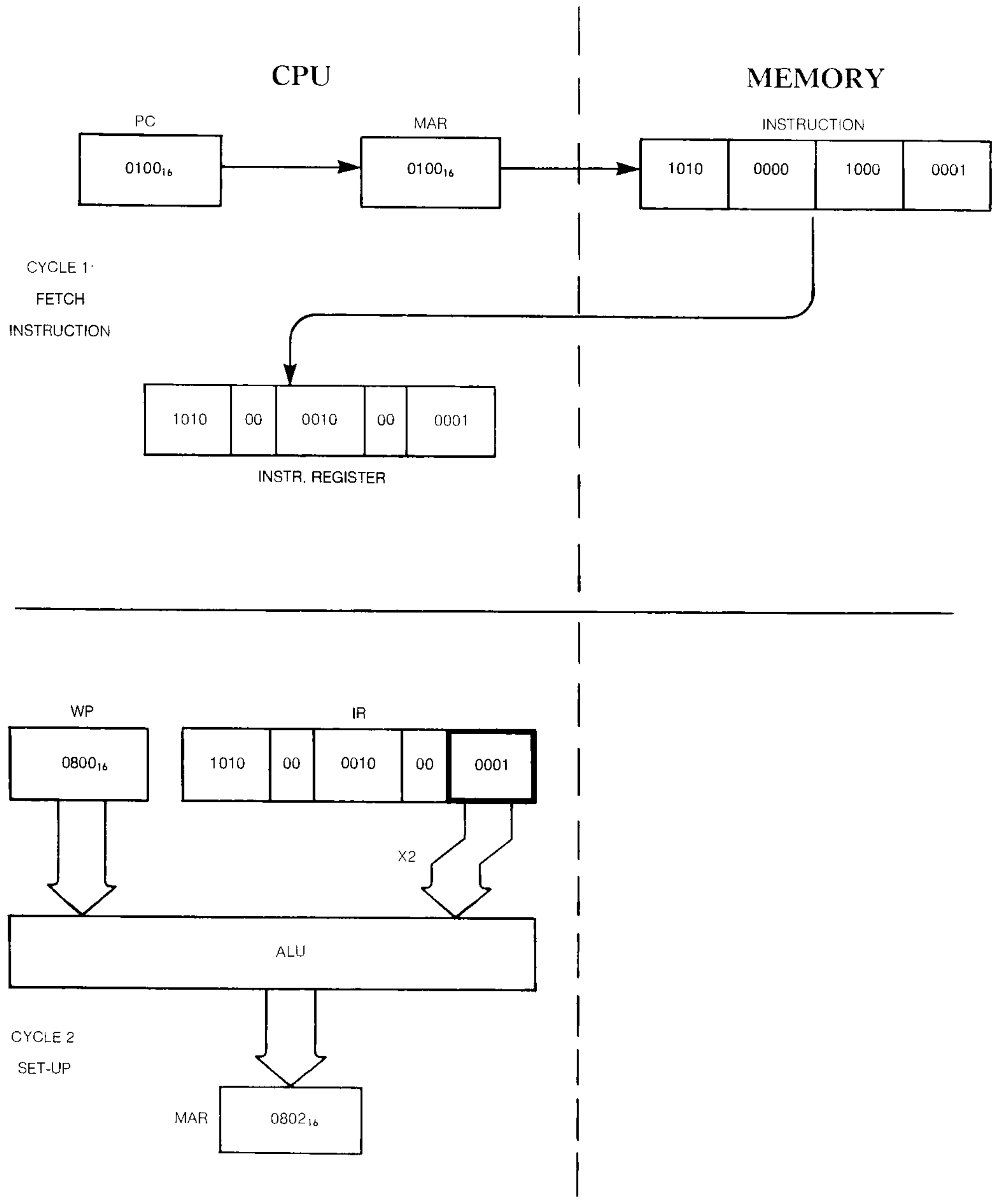


Figure 4-30a. Add Instruction Cycle

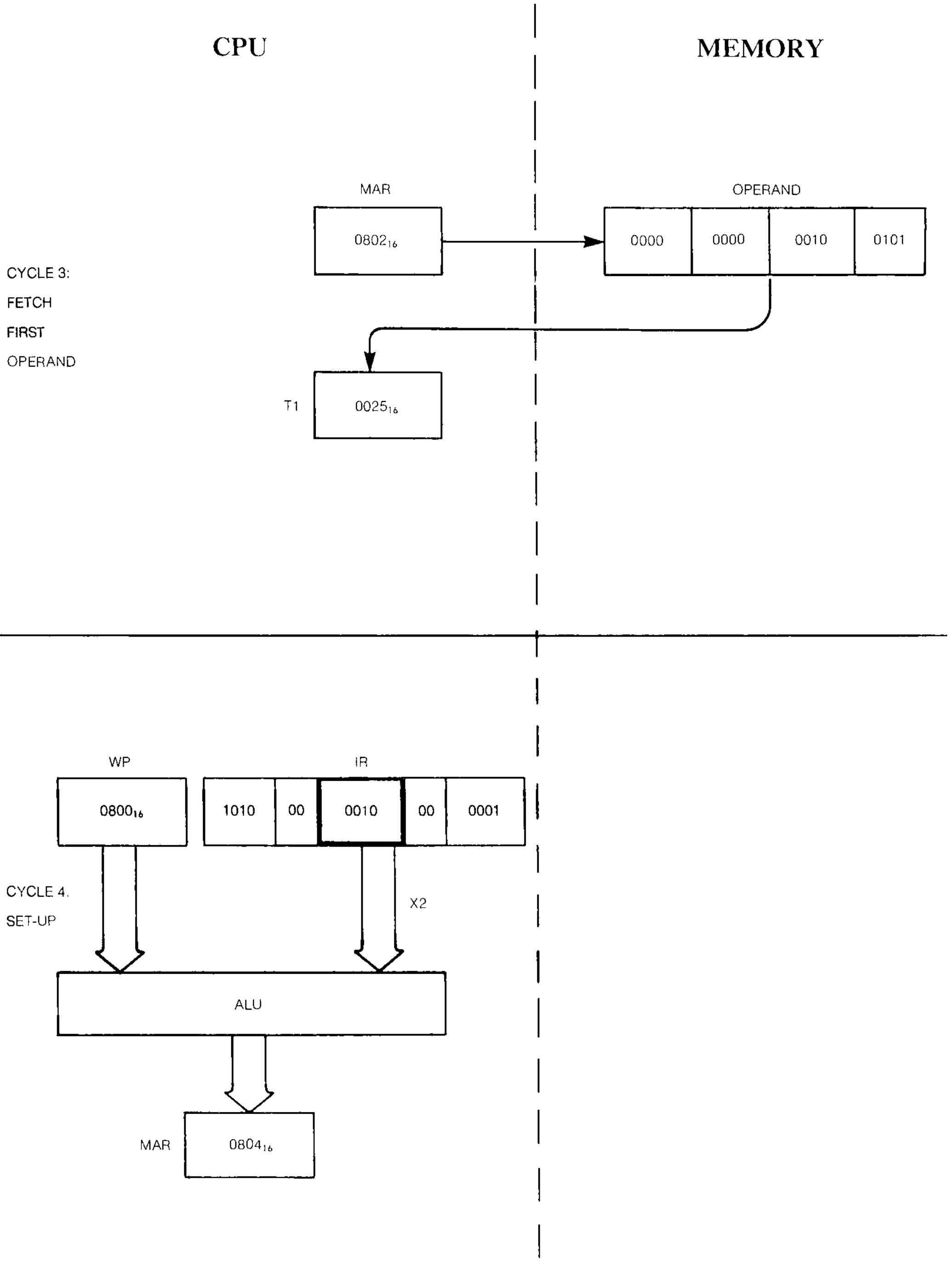


Figure 4-30b. Add Instruction Cycle

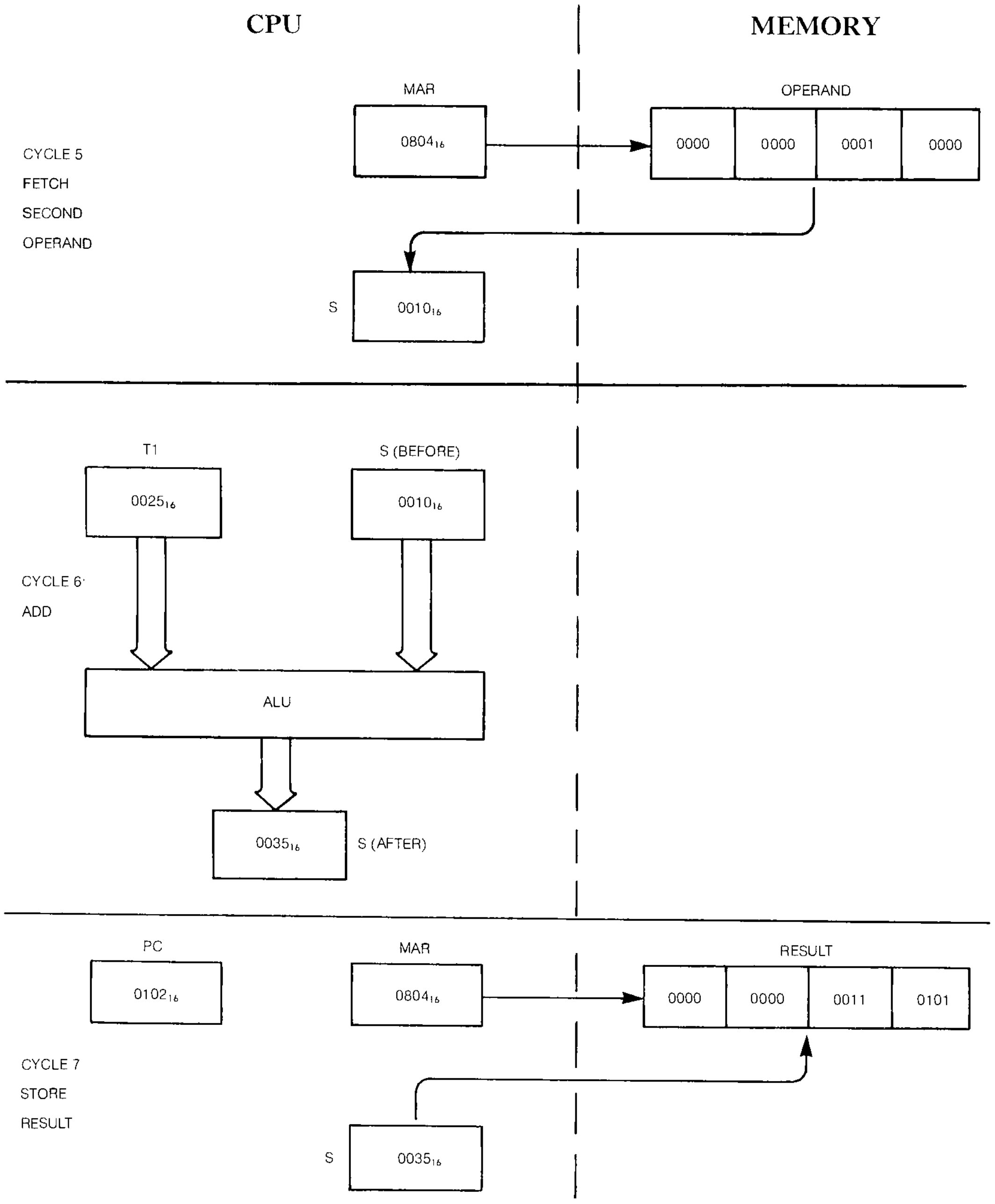


Figure 4-30c. Add Instruction Cycle

After all steps have been done, the processor checks to see if there is any pending interrupt operations to be performed and, if not, fetches the next instruction and the cycle continues. In the event that an interrupt signal were present, the processor would proceed to the appropriate interrupt service routine and continue execution from that point. Interrupts are described in detail in a special section of this chapter.

Each operation performed by the 9900 consists of a sequence of machine cycles. In each machine cycle the processor performs a data transfer with memory or with CRU and/or an arithmetic or logical operation internally with the ALU. A detailed discussion of the machine cycles for each instruction is included at the end of the chapter.

Each ALU machine cycle is two clock cycles long. In an ALU cycle no external data transfer occurs, but the ALU performs an arithmetic or logical operation on two operands contained internally. The function of the memory read cycle is to transfer a word of data contained in the memory to the processor. An ALU operation may be performed during the memory read cycle. Memory read cycles are a minimum of two clock cycles long. The memory write cycle is identical to the memory read cycle except that data is written rather than read from memory.

Each CRU output machine cycle is two clock cycles long. In addition to outputting a bit of CRU data, an ALU operation may also be performed internally. The CRU input cycle is identical to the CRU output cycle except that one bit of data is input rather than output.

Machine Cycle Limits

Table 4-1 lists information which will be useful for system design. The maximum number of consecutive memory-read cycles is used to calculate the maximum latency for the TMS 9900 to enter the hold state since the hold state is only entered from ALU, CRU input, or CRU output machine cycles. The minimum frequency of consecutive memory/ non-memory cycle sequences occurs when the DIV instruction is executed. This number is used to ensure that the refresh rate meets specifications when the transparent-refresh mode described in the memory section is used since memory is refreshed in this mode each time an ALU or CRU cycle follows a memory cycle. *Figure 4-31* shows the logic to generate a pulse for each memory access cycle. Consecutive cycle timing is shown in *Figure 4-32.*

Table 4-1. Machine Cycle Limits

	MINIMUM	MAXIMUM
Consecutive Memory Read Cycles	1	3
Consecutive Memory Write Cycles	1	1
Consecutive ALU Cycles	1	51
Consecutive CRU Cycles	1	16
Frequency of Consecutive memory/non-memory cycle pairs (used for transparent refresh)	5 pairs (64 machine cycles during DIV.)	

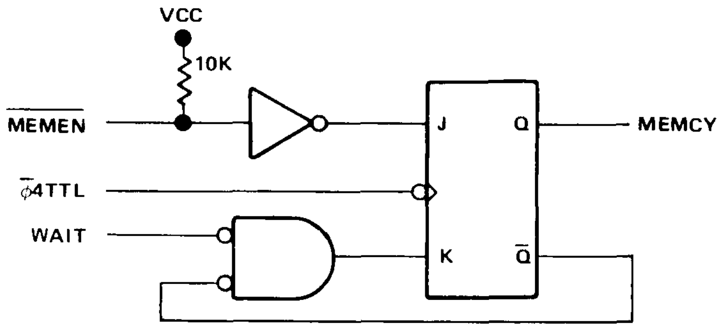


Figure 4-31. Memory Cycle Pulse Generation

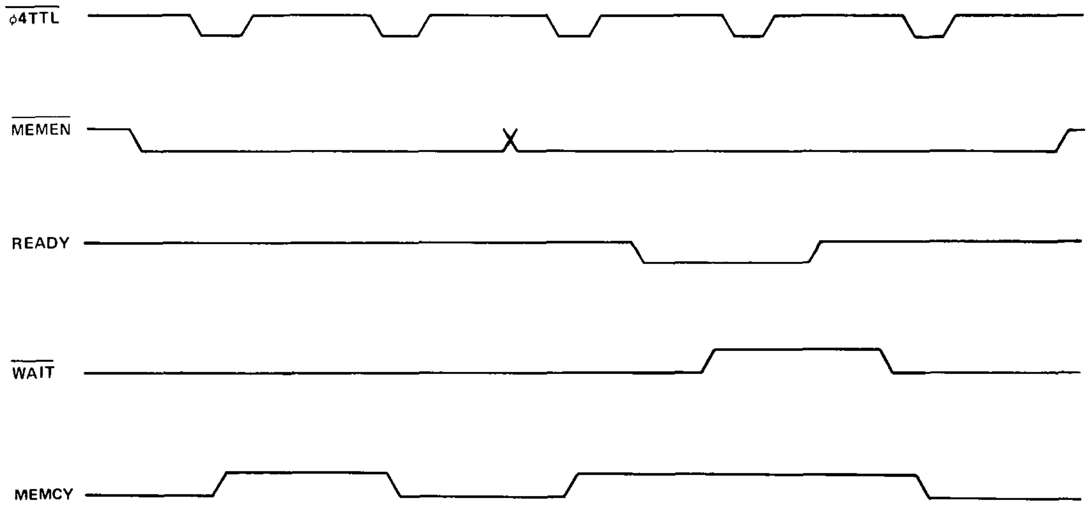
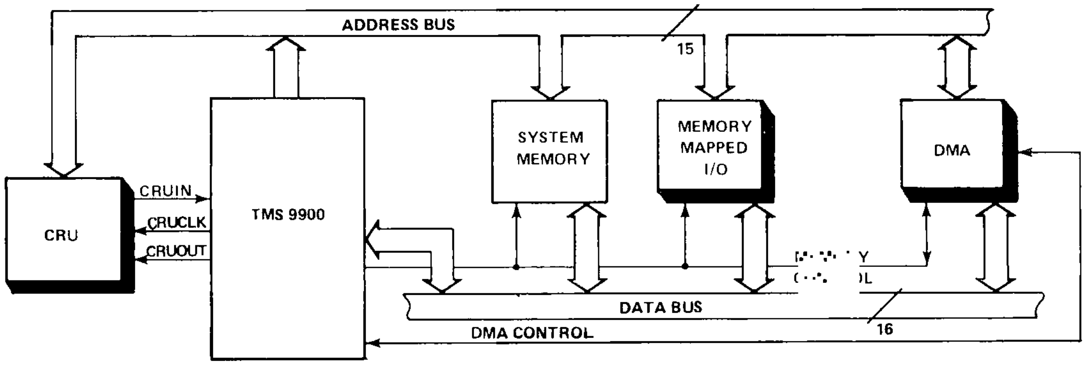


Figure 4-32. Memory Cycle Pulse Timing

INPUT/OUTPUT

The 9900 has three I/O modes: direct memory access (DMA), memory mapped, and communications register unit (CRU). This multi-mode capability enables the designer to optimize a 9900 I/O system to match a specific application. One or all modes can be used, as shown in *Figure 4-33.*



- COMMUNICATIONS REGISTER UNIT - CRU
- MEMORY MAPPED I/O
- DIRECT MEMORY ACCESS - DMA

Figure 4-33. 9900 I/O Capability

Direct Memory Access

DMA is used for high-speed block data transfer when CPU interaction is undesirable or not required. The DMA control circuitry can be relatively complex and expensive when compared to other I/O methods. However, a special interface device, the TMS 9911, is available for DMA control. This device is described in Chapter 8.

The 9900 controls CRU-based I/O transfers between the memory and peripheral devices. Data must pass through the CPU during these program-driven I/O transfers, and the CPU may need to be synchronized with the I/O device by interrupts or status-bit polling.

Some I/O devices, such as disk units, transfer large amounts of data to or from memory. Program driven I/O can require relatively large response times, high program overhead, or complex programming techniques. Consequently, direct memory access (DMA) is used to permit the I/O device to transfer data to or from memory without CPU intervention. DMA can result in a high I/O response time and system throughput, especially for block data transfers. The DMA control circuitry is somewhat more expensive and complex than the economical CRU I/O circuitry and should therefore be used only when required.

The 9900-based DMA can occur in the same modes as dynamic memory refresh: block, cycle stealing, or transparent. The transparent DMA mode is implemented similar to the refresh mode and must be synchronized with memory refresh cycles if dynamic memory is used. The block and cycle stealing modes, however, use the CPU $\overline{\text{HOLD}}$ capability and are more commonly used. The I/O device holds $\overline{\text{HOLD}}$ active (low) when a DMA transfer needs to occur. At the beginning of the next available non-memory cycle, the CPU enters the hold state and raises HOLDA to acknowledge the $\overline{\text{HOLD}}$ request. The maximum latency time between the hold request and the hold acknowledge is equal to three clock cycles plus three memory cycles. The minimum latency time is equal to one clock cycle. A 3-megahertz system with no wait cycles has a maximum hold latency of nine clock cycles or 3 microseconds and a minimum hold latency of one clock cycle or 0.3 microseconds.

When HOLDA goes high, the CPU address bus, data bus, DBIN, $\overline{\text{MEMEN}}$, and $\overline{\text{WE}}$ are in the high-impedance state to allow the I/O device to use the memory bus. The I/O device must then generate the proper address, data, and control signals and timing to transfer data to or from the memory as shown in *Figure 4-34*. Thus the DMA device has control of the memory bus when the TMS 9900 enters the hold state (HOLDA = 1), and may perform memory accesses without intervention by the microprocessor. Since DMA operations, in effect remove the 9900 from control while memory accesses are being performed, no further discussion is provided in this manual. Because the lines shown in *Figure 4-34* go into high impedance when HOLDA = 1, the DMA controller must force these signals to the proper levels. The I/O device can use the memory bus for one transfer (cycle-stealing mode) or for multiple transfers (block mode). At the end of the DMA transfer, the I/O device releases $\overline{\text{HOLD}}$ and normal CPU operation proceeds. The 9900 $\overline{\text{HOLD}}$ and HOLDA timing are shown in *Figure 4-35*.

Memory Mapped I/O

Memory mapped I/O permits I/O data to be addressed as memory with parallel data transfer through the system data bus. Memory mapped I/O requires a memory bus compatible interface; that is, the device is addressed in the same manner as a memory, thus the interface is identical to that of memory. *Figure 4-36* shows a memory mapped I/O interface with eight latched outputs and eight buffered inputs. In using memory mapped I/O for output only, care must be taken in developing the output device strobe to ensure it is not enabled during the initial read of the memory address, since the 9900 family of processors first reads, then writes data to a memory location in write operations. This can be effectively accomplished by using the processor write control signal $\overline{\text{WE}}$ in decoding the output address.

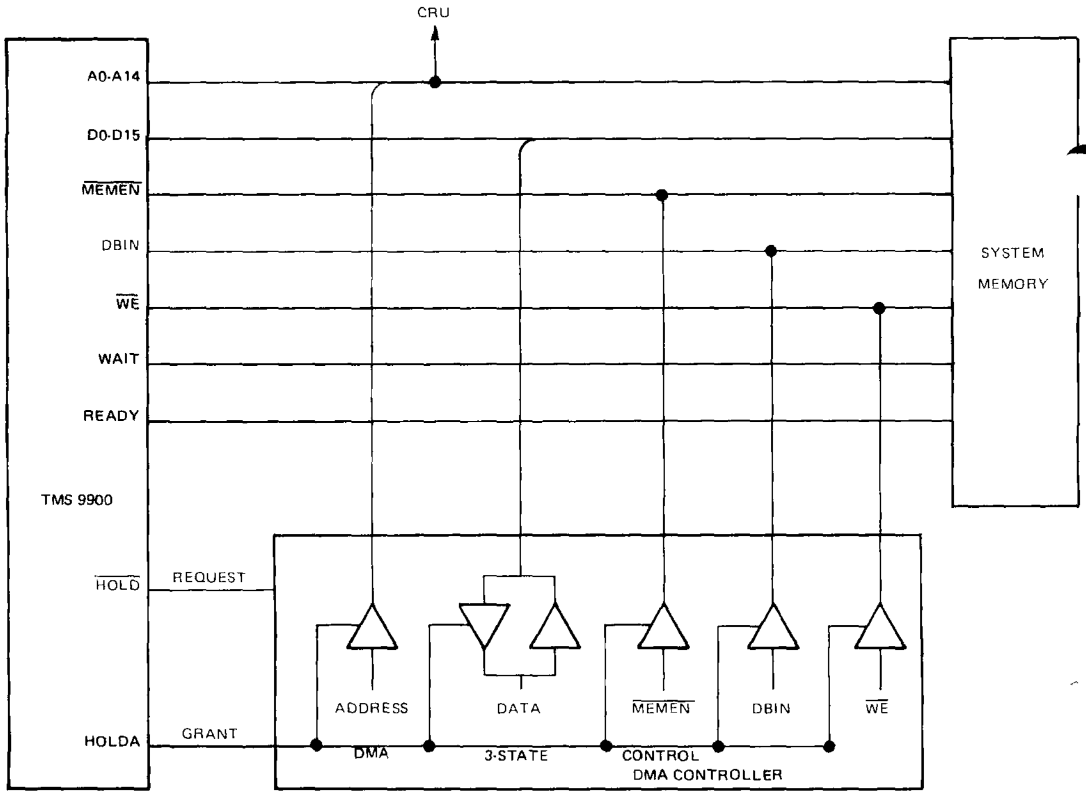


Figure 4-34. DMA Bus Control

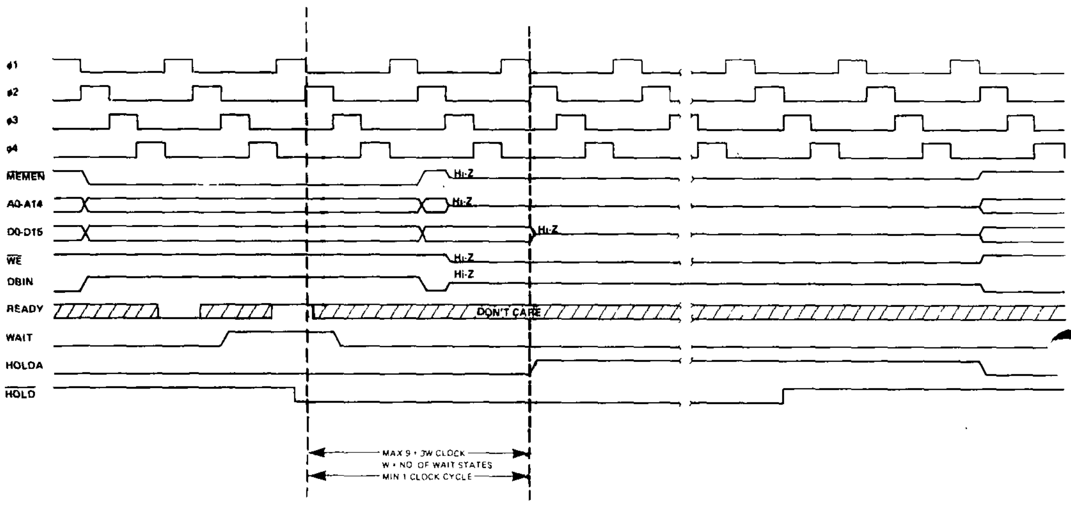


Figure 4-35. $\overline{\text{HOLD}}$ and HOLDA Timing

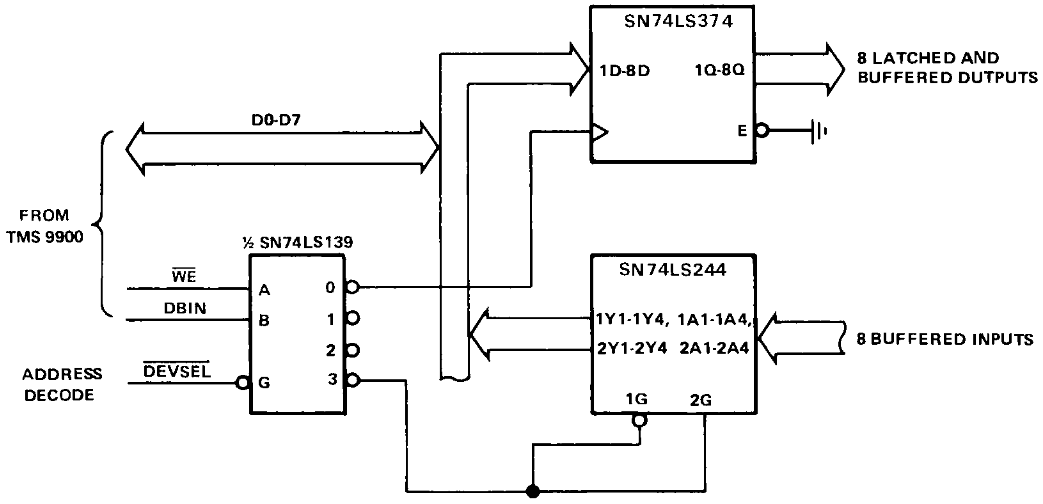


Figure 4-36. 8-Bit Memory Mapped I/O Interface

Communication Register Unit (CRU)

CRU I/O uses a dedicated bit addressable interface for I/O. The CRU instructions permit transfer of one to sixteen bits. The CRU interface requires fewer interface signals than the memory interface and can be expanded without affecting the memory system. In the majority of applications, CRU I/O is superior to memory mapped I/O as a result of the powerful bit manipulation capability, flexible field lengths, and simple bus structure.

The CRU bit manipulation instructions eliminate the masking instructions required to isolate a bit in memory mapped I/O. The CRU multiple-bit instructions allow the use of I/O fields not identical to the memory word size, thus permitting optimal use of the I/O interface. Therefore, the CRU minimizes the size and complexity of the I/O control programs, while increasing system throughput.

The CRU does not utilize the memory data bus. This can reduce the complexity of printed circuit board layouts for most systems. The standard 16-pin CRU I/O devices are less expensive and easier to insert than larger, specially designed, memory mapped I/O devices. The smaller I/O devices are possible as a result of the bit addressable CRU bus which eliminates the need for multiple pins dedicated to a parallel-data bus with multiple control lines. System costs are lower because of simplified circuit layouts, increased density, and lower component costs.

CRU Interface

The interface between the 9900 and CRU devices consists of address bus lines A0-A14, and the three control lines, CRUIN, CRUOUT, and CRUCLK as shown in *Figure 4-33.* A0-A2 indicate whether data is to be transferred and A3-A14 contain the address of the selected bit for data transfers; therefore, up to 2^{12} or 4,096 bits of input and 4,096 bits of output may be individually addressed. CRU operations and memory-data transfers both use A0-A14; however, these operations are performed independently, thus no conflict arises. The $\overline{\text{MEMEN}}$ line may be used to distinguish between CRU and memory cycles.

CRU Interface Logic

CRU based I/O interfaces are easily implemented using either CRU peripheral devices such as the TMS 9901 or the TMS 9902, or TTL multiplexers and addressable latches, such as the TIM 9905 (SN74LS251) and the TIM 9906 (SN74LS259). These I/O circuits can be easily cascaded with the addition of simple address decoding logic.

▸4 *TTL Outputs.* The TIM 9906 (SN74LS259) octal-addressable latch can be used for CRU outputs. The latch outputs are stable and are altered only when the CRUCLK is pulsed during a CRU output transfer. Each addressable latch is enabled only when addressed as determined by the upper address bits. The least-significant address bits (A12-A14) determine which of the eight outputs of the selected latch is to be set equal to CRUOUT during CRUCLK, and shown in *Figure 4-37.*

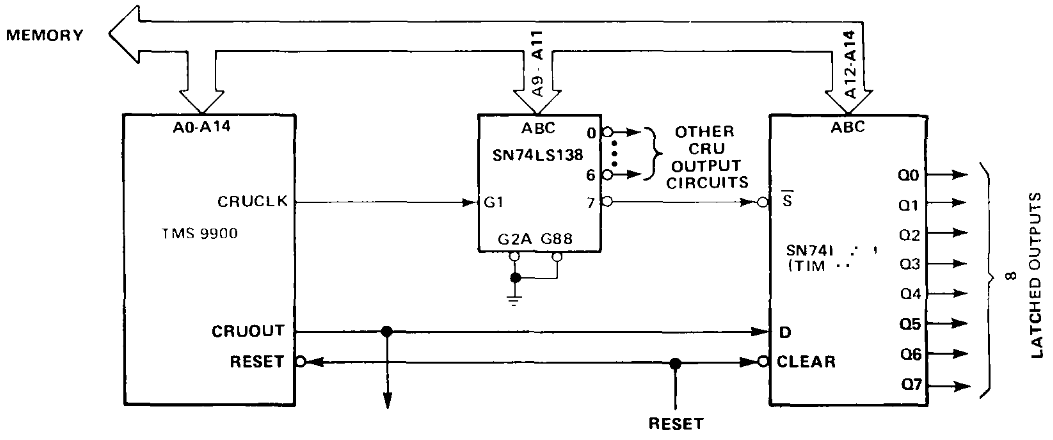


Figure 4-37. Latched CRU Interface

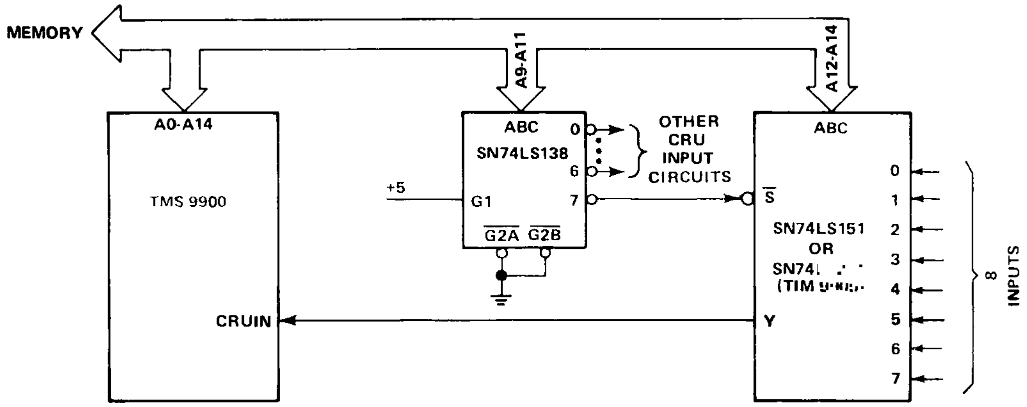


Figure 4-38. Multiplexer CRU Interface

TTL Inputs. The SN74LS151 and TIM 9905 (SN74LS251) octal multiplexers are used for CRU inputs as shown in *Figure 4-38.* The multiplexers are continuously enabled with CRUIN equal to the addressed input. The TIM 9905 should be used for larger systems since its three-state outputs permit simple "wire-ORing" of parallel-input multiplexers.

Expanding CRU I/O

A CRU interface with eight inputs and eight outputs is shown in *Figure 4-39* using the TMS 9901. An expanded interface with 16 inputs and 16 outputs is shown in *Figure 4-40* using TTL devices. The CRU inputs and outputs can be expanded up to 4096 inputs and 4096 outputs by decoding the complete CRU address. Larger I/O requirements can be satisfied by using memory mapped I/O or by using a CRU bank switch, which is set and reset under program control. When reset, the lower CRU I/O bank is selected, and when set, the upper CRU I/O bank is selected. In actual system applications, however, only the exact number of interface bits required need to be implemented. It is not necessary to have a 16-bit CRU output register to interface a 10-bit device.

CRU Machine Cycles

Each CRU operation consists of one or more CRU output or CRU input machine cycles, each of which is two clock cycles long. As shown in *Table 4-2,* five instructions (LDCR, STCR, SBO, SBZ, TB) transfer data to or from the 9900 with CRU machine cycles, and five external control instructions (IDLE, RSET, CKOF, CKON, LREX) generate control signals with CRU output machine cycles.

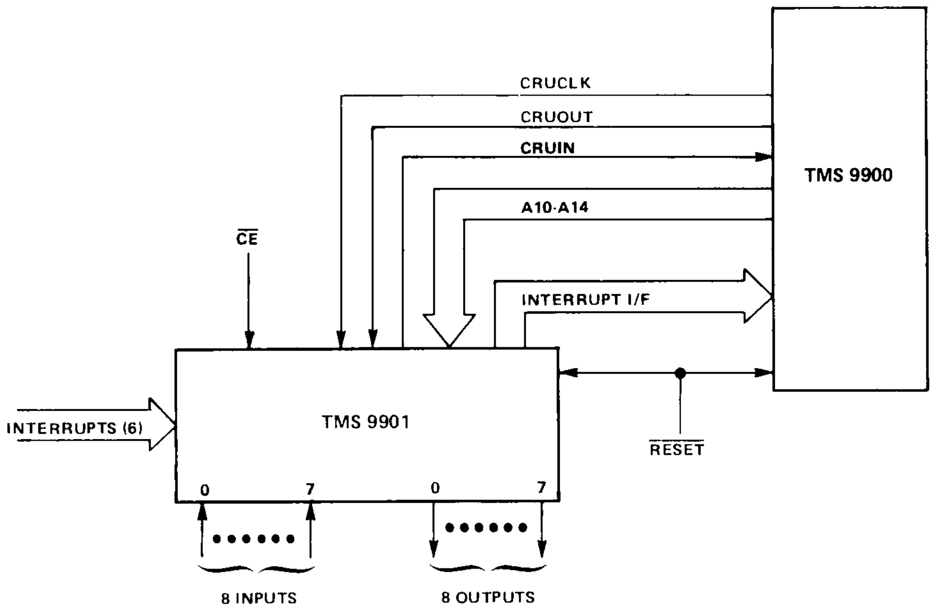


Figure 4-39. 8-Bit CRU Interface

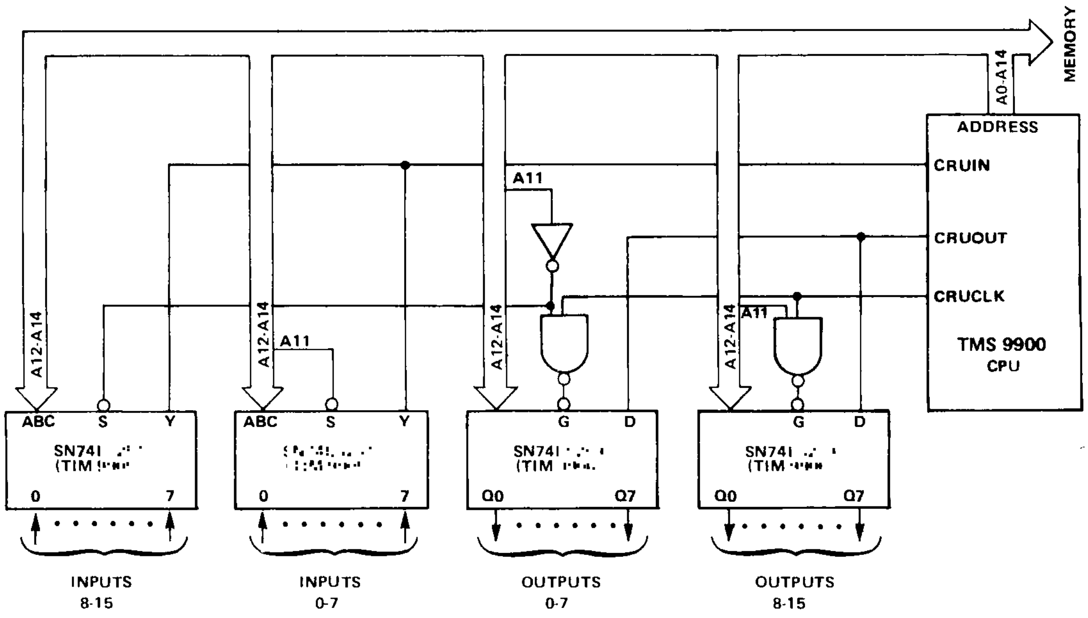


Figure 4-40. 16-Bit CRU Interface

Table 4-2. Instructions Generating CRU Cycles

INSTRUCTION	NUMBER OF CRU CYCLES	TYPE OF CRU CYCLES	A0-A2	DATA TRANSFER
LDCR	1-16	Output	0 0 0	Yes
STCR	1-16	Input	0 0 0	Yes
SBO	1	Output	0 0 0	Yes
SBZ	1	Output	0 0 0	Yes
TB	1	Input	0 0 0	Yes
IDLE	1	Output	0 1 0	No
RSET	1	Output	0 1 1	No
CKOF	1	Output	1 0 1	No
CKON	1	Output	1 1 0	No
LREX	1	Ouput	1 1 1	No

Figure 4-41 shows the timing for CRU *output* machine cycles. Address (A0-A14) and data (CRUOUT) are output on $\phi 2$ of clock cycle 1. One clock cycle later, the 9900 outputs a pulse on CRUCLK for ½ clock cycle. Thus, CRUCLK can be used as a strobe, since address and data are stable during the pulse. Referring again to *Table 4-2,* it is important to note that output data is transferred only when A0-A2 = 000. Otherwise, no data transfer should occur, and A0-A2 should be decoded to determine which external control instruction is being executed. These external control instructions may be used to perform simple control operations. The generation of control strobes for external instructions and a data transfer strobe (OUTCLK) is illustrated in *Figure 4-42.* If none of the external control instructions is used, A0-A2 need not be decoded for data transfer since they will always equal 000.

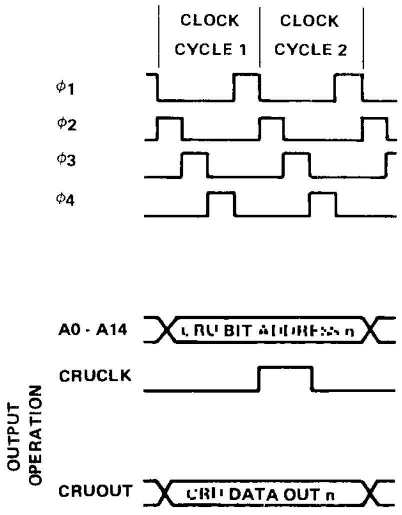
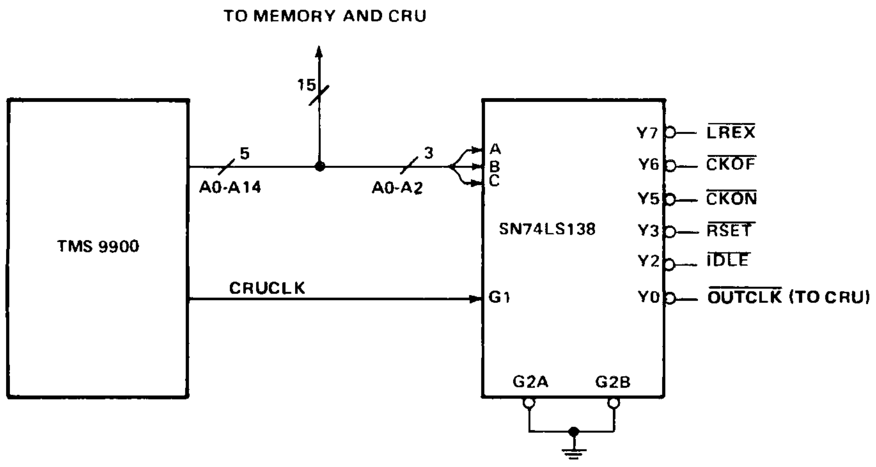


Figure 4-41. CRU Output Machine Cycle Timing



4

Figure 4-42. CRU Control Strobe Generation

The timing for CRU *input* machine cycles is shown in *Figure 4-43.* The address is output at the beginning of the first clock cycle. The CRUIN data input is sampled on ϕ1 of clock cycle 2. Thus, CRU input is accomplished by simply multiplexing the addressed bit onto the CRUIN input. A0-A2 will always be 000, and may be ignored. CRU input machine cycles cannot be differentiated from ALU cycles by external logic, thus no operations (such as clearing interrupts) other than CRU input should be performed during CRU input machine cycles.

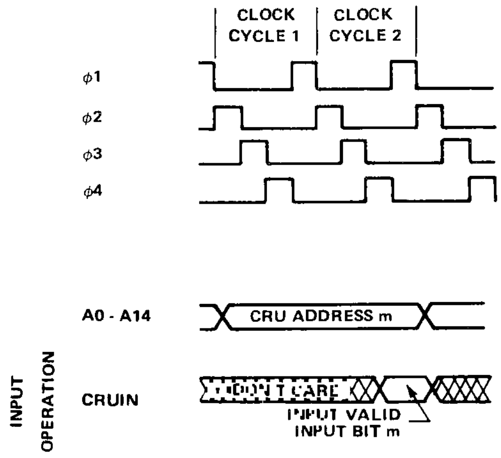


Figure 4-43. CRU Input Machine Cycle Timing

CRU Data Transfer

In order to transfer data from a memory location to an external latch in the Communications Register Unit, or to transfer data from a CRU multiplexer to memory, special instructions must be used. The CRU instructions are:

SBO	Set bit to one (output)
SBZ	Set bit to zero (output)
TB	Test bit (input)
LDCR	Load n bits to CRU (output)
STCR	Receive n bits from CRU (input)

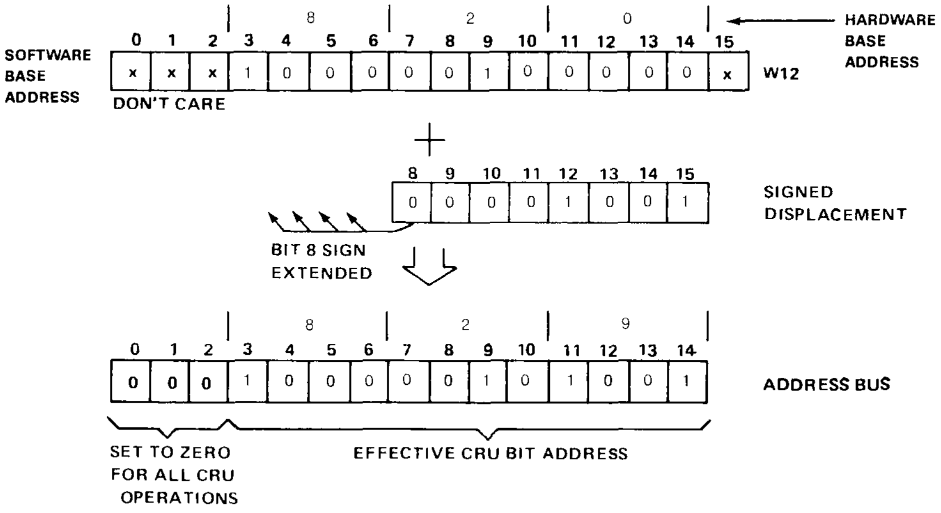
These instructions always use the address bus to identify the bit or bits to be transferred, but they make the actual transfer of data over the dedicated CRU lines, CRUIN and CRUOUT. Addressing of the CRU bits is accomplished by adding a portion of the instruction word to a CRU base address register. The use of such a base address technique allows one program segment to service any number of identical I/O devices. For example: five TMS 9902's each with its own assigned base address can be operated from a single program, provided the base address register is properly set at the beginning. In the 9900, workspace register 12 is the CRU software base address register. All CRU instructions use the contents of this register in addressing individual CRU bits.

The CRU hardware base address is defined by bits 3-14 of the current WR12 when CRU data transfer is performed. Bits 0-2 and bit 15 of WR12 are ignored for CRU address determination.

For single-bit CRU instructions (SBO, SBZ, TB), the address of the CRU bit to or from which data is transferred is determined as shown in *Figure 4-44.* Bits 8-15 of the machine code instruction contain a signed displacement. This signed displacement is added to the CRU hardware base address (bits 3-14 of WR12). The result of this addition is output on A3-A14 during the CRU output or the CRU input machine cycle.

For example, assume the instruction "SBO 9" is executed when WR12 contains a value of 1040_{16}. The machine code for "SBO 9" is $1D09_{16}$ and the signed displacement is 0009_{16}. The CRU hardware base address is 0820_{16} (bits 0-2 and bit 15 are ignored). Thus, the effective CRU bit address is $0820_{16} + 0009_{16} = 0829_{16}$, and this value is output on A0-A14 during the CRU output machine cycle.

As a second example, assume that the instruction TB −32 is executed when $WR12 = 100_{16}$. The effective CRU address is 80_{16}. (CRU hardware base) + $FFEO_{16}$ (signed displacement) = 60_{16}. Thus, the TB −32 instruction in this example causes the value of the CRU input bit at address 60_{16} to be transferred to bit 2 of the status register. This bit is tested in the execution of the JEQ or JNE instructions; if it is a one, the PC will be loaded with a new value (JEQ instruction).



▶4

Figure 4-44. TMS 9900 Single-Bit CRU Address Development

LDCR Instruction

The LDCR may transfer from 1 to 16 fits of output data with each instruction. Output of each bit is performed by a CRU output machine cycle; thus, the number of CRU output machine cycles performed by an LDCR instruction is equal to the number of bits to be transferred.

As an example, assume that the instruction "LDCR @600,10" is executed, and that WR12 = 800_{16} and the memory word at address 600 contains the bit pattern shown in *Figure 4-45.* In the first CRU output machine cycle the least significant bit of the operand (a) is output on CRUOUT. In each successive machine cycle the address is incremented by one and the next least-significant bit of the operand is output on CRUOUT, until 10 bits have been output. It is important to note that the CRU base address is unaltered by the LDCR instruction, even though the address is incremented as each successive bit is output.

STCR Instruction

The STCR instruction causes from 1 to 16 bits of CRU data to be transferred into memory. Each bit is input by a CRU input machine cycle.

Consider the circuit shown in *Figure 4-46.* The CRU interface logic multiplexes input signals m-t onto the CRUIN line for addresses 200_{16}-207_{16}. If WR12 = 400_{16} when the instruction "STCR @ 602,6" is executed, the operation is performed as shown in *Figure 4-47.* At the end of the instruction, the six LSBs of memory byte 602 are loaded with m-r. The upper bits of the operand are forced to zero.

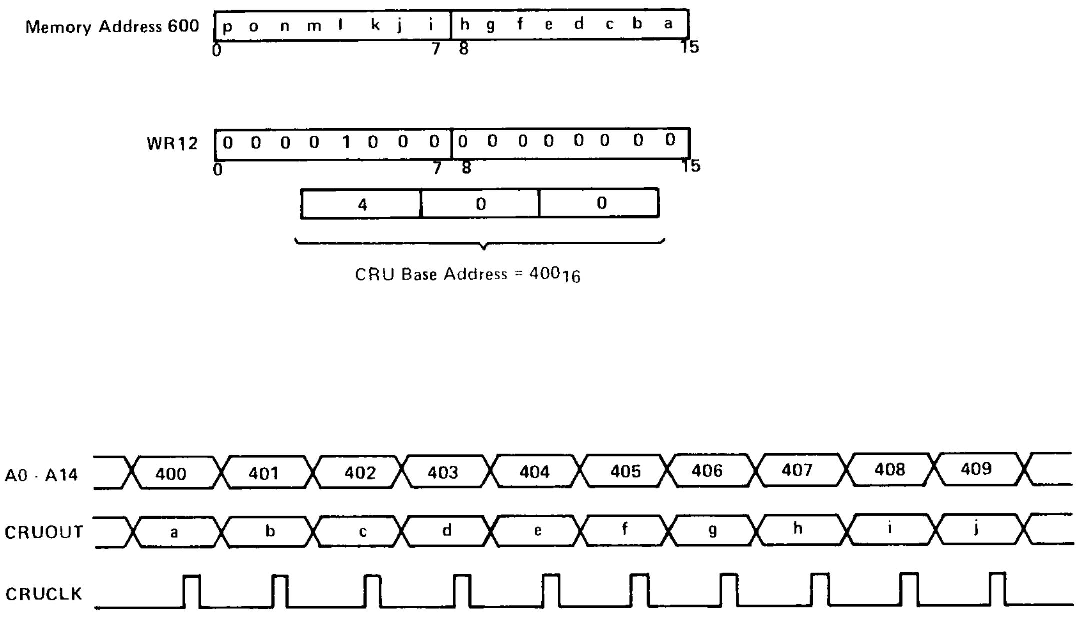


Figure 4-45. Multiple-Bit CRU Output

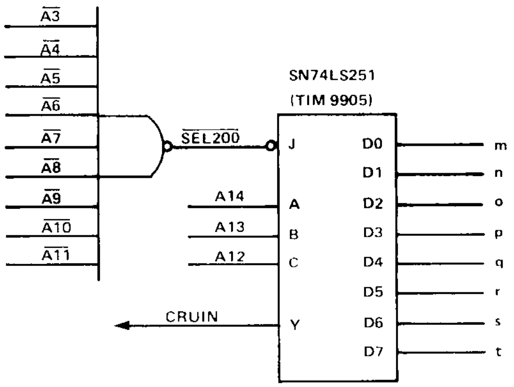


Figure 4-46. Example CRU Input Circuit

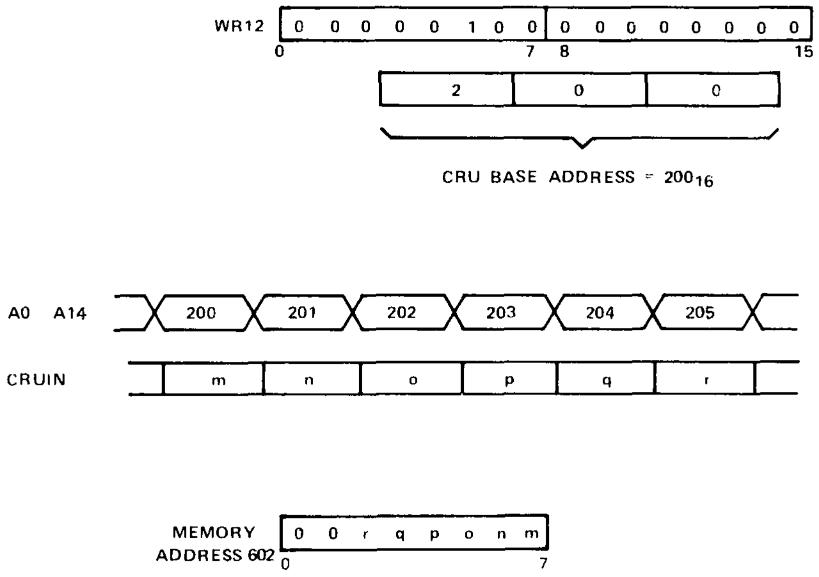


Figure 4-47. Multiple-Bit CRU Input

CRU Paper Tape Reader Interface

CRU interface circuits are used to interface data and control lines from external devices to the 9900. This section describes an example interface from a paper tape reader. The paper tape reader is assumed to have the following characteristics:

1. It generates a TTL-level active-high signal (SPROCKET HOLE) on detection of a sprocket hole on the paper tape.
2. It generates an 8-bit TTL active-low data which stays valid during SPROCKET HOLE = 1.
3. It responds to a TTL-level active-high command (Paper Tape RUN) signal by turning on when PTRUN = 1 and turning off when PTRUN = 0.

Figure 4-48 illustrates the circuitry to interface the reader to the CRU. The interface is selected when $\overline{\text{PTRSEL}} = 0$; $\overline{\text{PTRSEL}}$ is decoded from the A0-A11 address outputs from the 9900. Thus, the output of the SN74LS251 is active only when $\overline{\text{PTRSEL}} = 0$; otherwise, the output is in high impedance and other devices may drive CRUIN. The data inputs are selected by A12-A14 and inverted, resulting in active high data input on CRUIN. The positive transition of SPROCKET HOLE causes $\overline{\text{PTRINT}}$ to go low. $\overline{\text{PTRINT}}$ is the active low interrupt from the interface. $\overline{\text{PTRINT}}$ is set high, clearing the interrupt, whenever a CRU output machine cycle is executed and the address causes $\overline{\text{PTRSEL}}$ to be active. When a one is output, $\overline{\text{PTRUN}}$ is set, enabling the reader, and the reader is disabled when a zero is output to the device. Thus, any time $\overline{\text{PTRUN}}$ is set or reset, the interrupt is automatically cleared.

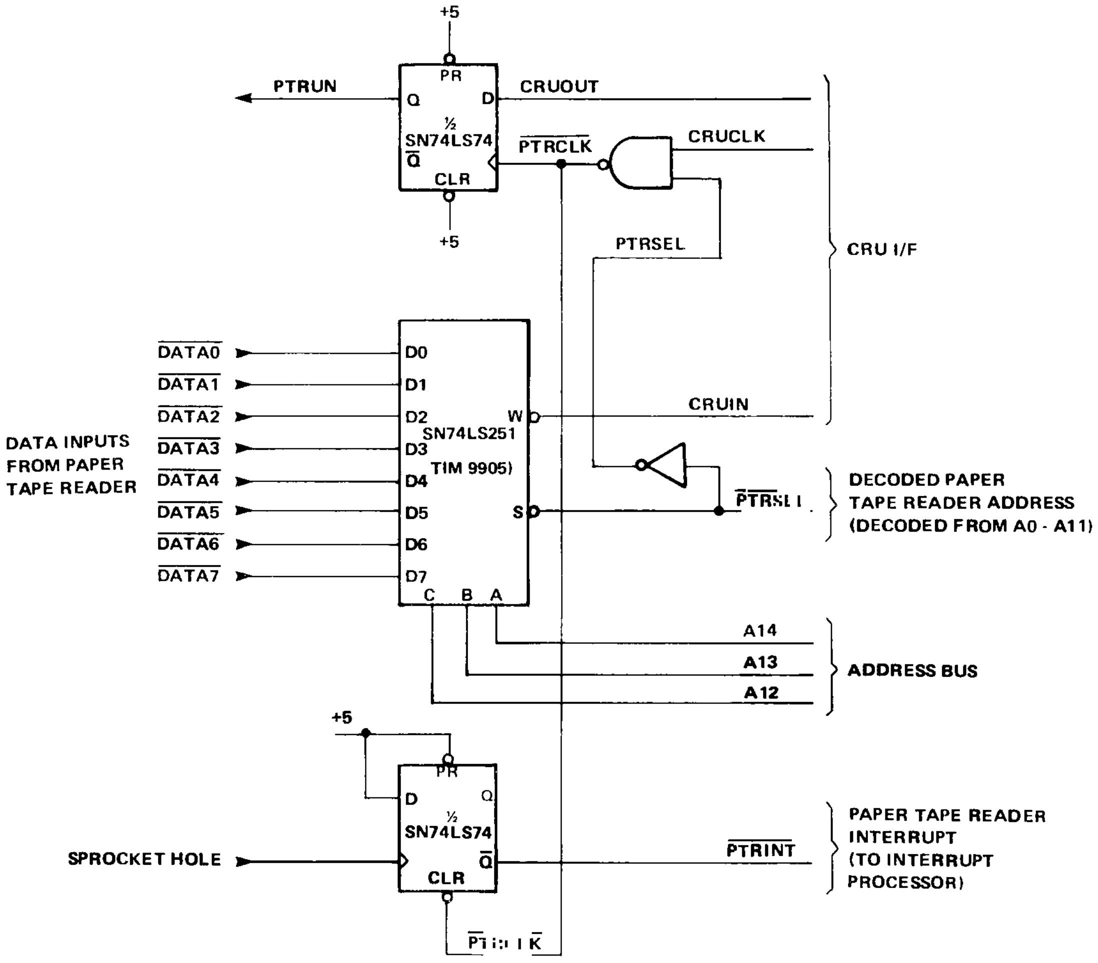


Figure 4-48. Paper Tape Reader Interface

The software routine in *Figure 4-49* controls the paper-tape reader interface described above. It is a re-entrant procedure that can be shared by several readers. The assumptions are that:

1. Each reader has its own workspace which is set up on the trap location for that reader's interrupt.
2. The workspace registers are allocated as shown in *Figure 4-50.*
3. The CRU input bits 0-7 (relative to CRU base) are reader data. CRU output bit 0 controls PTRUN and clears the interrupt.
4. The most significant byte of R9 = End of File Code.
5. R10 = Overflow Count
6. R11 = Data Table Pointer Address.

The procedure has two entry points. It is entered by a calling routine at PTRBEG to start the reader and it returns control to that routine. It is entered at PTRINT via interrupt to read a character. The return in this case is to the interrupted program.

The control program may be used by any number of paper-tape reader interfaces, as long as each interface has a separate interrupt level and workspace. As each reader issues an interrupt, the 9900 will process the interrupt beginning at location PTRINT. However, the workspace unique to the interrupting device is used. The organization of memory to control two paper tape readers is shown in *Figure 4-50.* The interrupt-transfer vector causes the appropriate WP value to be loaded. In both cases PTRINT, the entry point for the control program, is loaded into the PC.

```
PTRINT      STCR    *R11,8
            CB      *R11+, R9
            JEQ     PTREND
            DEC     R10
            JEQ     PTREND
PTRBEG      SBO     PTRUN
            RTWP
PTREND      SBZ     PTRUN
            LI      R10, MAXCOUNT
            RTWP
```

Figure 4-49. Paper Tape Reader Control Program

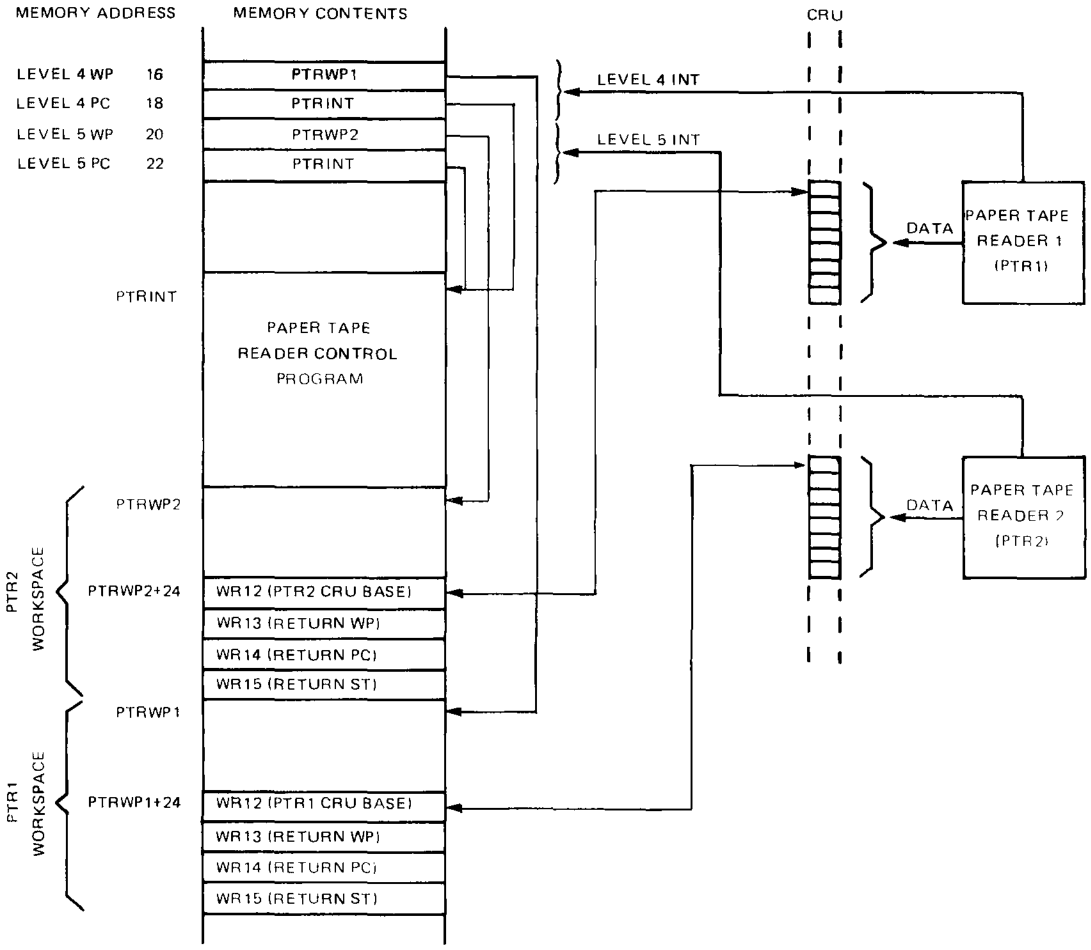


Figure 4-50. Software Configuration for Two Paper Tape Readers with Common Control Program

Burroughs SELF-SCAN Display Interface

This section describes a TMS9900 CRU interface to a Burroughs SELF-SCAN® panel display model SS30132-0070. The display panel has a 32-position, single-row character array with a repertoire of 128 characters.

The panel display operates in a serial-shift mode in which characters are shifted into the panel one at a time. Characters are shifted in right-to-left and can be shifted or backspaced left-to-right. A clear pulse erases the display.

The CRU display interface is shown in *Figure 4-51* and a display control subroutine is shown in *Figure 4-52.* The subroutine is called by one of two XOP instructions, XOP0 and XOP1. The calling routine passes the address and length of the output string in registers 8 and 9 of its workspace. The two XOP subroutines share the same workspace and perform the same function except that XOP1 clears the panel display first. The backspace feature is not used. The panel display is blanked during character entry.

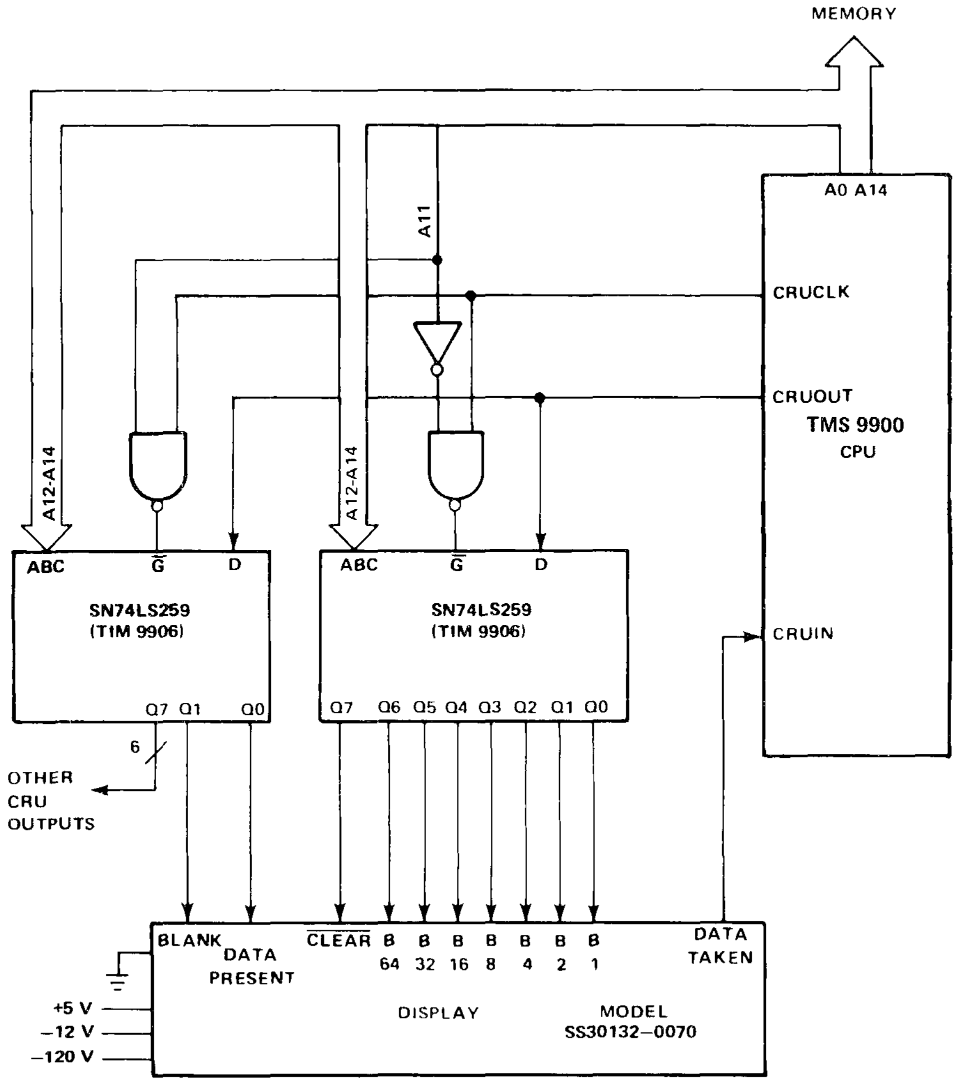


Figure 4-51. Display Control Interface

```
EIGHT     EQU     16
NINE      EQU     18

RXOP1     SBZ     7                   Clear Panel
          LI      R1,11

LOOP1     DEC     R1                  Delay >67µsec
          JNE     LOOP1
          SBO     7
RXOP2     SBO     9                   Blank Panel
          MOV     @EIGHT (13),1       Load Address (Old R8→R1)
          MOV     @NINE (13),2        Load Length (Old R9→R2)

LOOP2     LDCR    *1+,7               Output Char
          SBO     8                   Data Present

WAIT      TB      0                   Wait for Data Taken
          JEQ     Wait
          SBZ     8
          DEC     2                   Decrement Count
          JNE     LOOP2               Loop Until Through
          SBZ     9                   Unblank Panel
          RTWP                        Return
```

Figure 4-52. Burroughs SELF-SCAN® Display Control Program

INTERRUPTS

The TMS 9900 provides fifteen maskable interrupt levels in addition to the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions. The CPU has a priority ranking system to resolve conflicts between simultaneous interrupts and a level mask to disable lower priority interrupts. Once an interrupt is recognized, the CPU performs a vectored context switch to the interrupt service routine. The $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ functions are initiated by external input signals.

RESET

The $\overline{\text{RESET}}$ signal is normally used to initialize the CPU following a power-up. When active (low), the $\overline{\text{RESET}}$ signal inhibits $\overline{\text{WE}}$ and CRUCLK, places the CPU memory bus and control signals in a high-impedance state, and resets the CPU. When the $\overline{\text{RESET}}$ signal is released, the CPU fetches the restart vector from locations 0000 and 0002, stores the old WP, PC, and ST into the new workspace, resets all status bits to zero and starts execution at the new PC. The $\overline{\text{RESET}}$ signal must be held active for a minimum of three clock cycles. The $\overline{\text{RESET}}$ machine cycle sequence is shown in *Figure 4-53.*

A convenient method of generating the $\overline{\text{RESET}}$ signal is to use the Schmitt-triggered D-input of the TIM9904 clock generator. An RC network connected to the D-input maintains an active $\overline{\text{RESET}}$ signal for a short time immediately following the power-on, as shown in *Figure 4-54.*

CYCLE	TYPE	FUNCTION
*	*	Loop While Reset is Active
1	ALU	Set Up
2	ALU	Set Up
3	Memory	Fetch New WP, Move Status To T Reg, Clear Status
4	ALU	Set Up
5	Memory	Store Status
6	ALU	Set Up
7	Memory	Store PC
8	ALU	Set Up
9	Memory	Store WP
10	ALU	Set Up
11	Memory	Fetch New PC
12	ALU	Set Up MAR for Next Instruction

Figure 4-53. $\overline{\text{RESET}}$ Machine Cycles

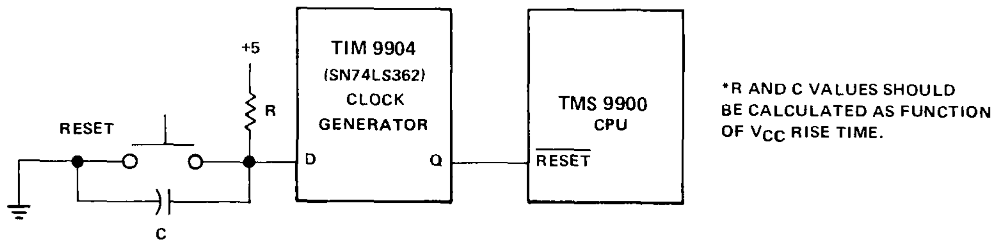

+5
R
RESET
C
TIM 9904
(SN74LS362)
CLOCK
GENERATOR
D
Q
TMS 9900
CPU
$\overline{\text{RESET}}$
*R AND C VALUES SHOULD BE CALCULATED AS FUNCTION OF V_{CC} RISE TIME.

Figure 4-54. $\overline{\text{RESET}}$ Generation

LOAD

The $\overline{\text{LOAD}}$ signal is normally used to implement a restart ROM loader or front panel functions. When active (low), the $\overline{\text{LOAD}}$ signal causes the CPU to perform a non-maskable interrupt. The $\overline{\text{LOAD}}$ signal can be used to terminate a CPU idle state.

The $\overline{\text{LOAD}}$ signal should be active for one instruction period. Since there is no standard TMS 9900 instruction period, IAQ should be used to determine instruction boundaries. If the $\overline{\text{LOAD}}$ signal is active during the time that the $\overline{\text{RESET}}$ signal is released, the CPU will perform the $\overline{\text{LOAD}}$ function immediately after the $\overline{\text{RESET}}$ function is completed. The CPU performs the $\overline{\text{LOAD}}$ function by fetching the $\overline{\text{LOAD}}$ vector from addresses $FFFC_{16}$ and $FFFE_{16}$, storing the old WP, PC, and ST in the new workspace, and starting the $\overline{\text{LOAD}}$ service routine at the new PC, as shown in *Figure 4-55.*

An example of the use of the $\overline{\text{LOAD}}$ signal is a bootstrap ROM loader. When the $\overline{\text{LOAD}}$ signal is enabled, the CPU enters the service routine, transfers a program from peripheral storage to RAM, and then transfers control to the loaded program.

Figure 4-56 illustrates the generation of the $\overline{\text{LOAD}}$ signal for one instruction period.

CYCLE	TYPE	FUNCTION
1	ALU	Set Up
2	Memory Read	Fetch New WP
3	ALU	Set Up
4	Memory Write	Store Status
5	ALU	Set Up
6	Memory Write	Store PC
7	ALU	Set Up
8	Memory Write	Store WP
9	ALU	Set Up
10	Memory Read	Fetch New PC
11	ALU	Set UP MAR for Next Instruction

Figure 4-55. $\overline{\text{LOAD}}$ Machine Cycle Sequence

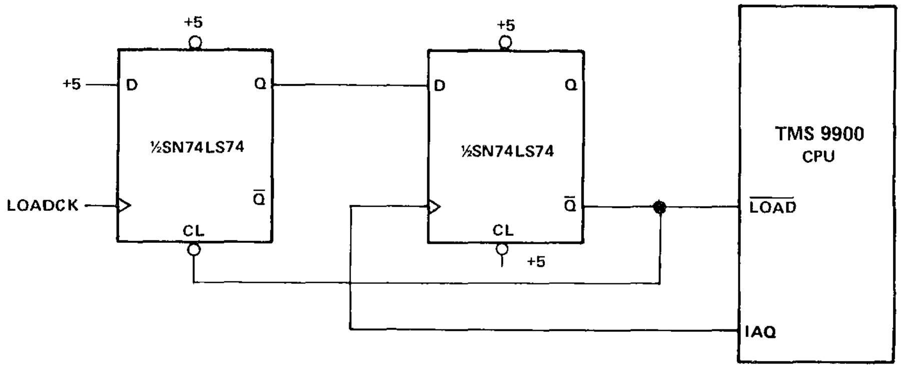


Figure 4-56. $\overline{\text{LOAD}}$ Generation

Basic Machine Cycle

The interrelationship between the $\overline{\text{LOAD}}$ and $\overline{\text{RESET}}$ signals and the general operation of the 9900 and execution of instructions may best be shown by the flow diagram in *Figure 4-57.* An orderly starting procedure involves the holding of the $\overline{\text{RESET}}$ line low when power is applied to the chip. After application of power and after the clock has begun to run, the internal instruction control circuitry checks to see if the $\overline{\text{RESET}}$ line is held low. and, if the answer is "yes", will stay in a loop as shown in the diagram. When the $\overline{\text{RESET}}$ line goes high, it is no longer active and a level zero interrupt is taken in which the $\overline{\text{RESET}}$ vector, the numbers to fill the workspace pointer and program counter registers, are fetched from memory locations zero and two. Furthermore, the previous values of the workspace pointer, program counter and status register are stored in the new workspace, although these values are random numbers immediately following power up. Following this, the interrupt mask is set to zero to mask all other interrupts.

The next decision is regarding the $\overline{\text{LOAD}}$ line. If this particular line is active, or low, then immediately there will be another context switch in which the $\overline{\text{LOAD}}$ vector will be brought in from the last two locations in memory, FFFC_{16} and FFFE_{16}, and loaded into the workspace pointer and program counter respectively. If the $\overline{\text{LOAD}}$ is not active, the 9900 proceeds directly to an instruction acquisition cycle. In either case, the very next step is to fetch the instruction from the memory and execute it.

Following this, the program counter is updated and a sequence of checks made regarding the $\overline{\text{LOAD}}$, XOP, and interrupt conditions. First is the check for the $\overline{\text{LOAD}}$ line. If this is active, the $\overline{\text{LOAD}}$ context switch will occur. If not, there will be a test to see if the instruction just executed was an XOP or BLWP. If not, the interrupt request line will be checked. If there is not an interrupt request, and the last instruction was not an idle instruction, the machine may proceed to fetch the next instruction and continue.

In the event that the last instruction executed was an XOP or BLWP, the 9900 will ignore the interrupt request line and will proceed to fetch a new instruction. This insures that at least one instruction of a subprogram that is entered via a context switch will be executed before another context switch may occur, such as an interrupt. In the event that the interrupt request line is active following the execution of a normal instruction, a test is made to determine that the interrupt is valid, that is to say, "Is the interrupt mask set to allow this interrupt." If the interrupt is not allowed, the processor proceeds to fetch the next instruction. In the event that it is allowed, a context switch will be made and the interrupt vector from the appropriate locations in the first 32 words of memory will be fetched and the workspace pointer and program counter will be loaded with the new numbers. As a part of this context switch, the interrupt mask is set to a level one less than the interrupt just taken. This is to insure that no lower priority interrupt may occur during the servicing of the current interrupt cycle. Notice further that in this diagram that the logic is such that at least one instruction of any subprogram will be executed immediately following a context switch. The only exception to this is the simultaneous presence of $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ signals. Finally, the idle instruction will suspend instruction execution in the 9900 until an interrupt, $\overline{\text{RESET}}$ or $\overline{\text{LOAD}}$ signal occurs.

Maskable Interrupts

The TMS 9900 has 16 interrupt levels with the lower 15 priority levels used for maskable interrupts. The maskable interrupts are prioritized and have transfer vectors similar to the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ vectors.

Interrupt Service

A pending interrupt of unmasked priority level is serviced at the end of the current instruction cycle with two exceptions. The first instruction of a $\overline{\text{RESET}}$, $\overline{\text{LOAD}}$, or interrupt service routine is executed before the CPU tests the $\overline{\text{INTREQ}}$ signal. The interrupt is also inhibited for one instruction if the current instruction is a branch and load workspace pointer instruction (BLWP) or an extended operation (XOP). The one instruction delay permits one instruction to be completed before an interrupt context switch can occur. A LIMI instruction can be used as the first instruction in a routine to lock out higher priority maskable interrupts.

The pending interrupt request should remain active until recognized by the CPU during the service routine. The interrupt request should then be cleared under program control. The CRU bit manipulation instructions can be used to recognize and clear the interrupt request.

The interrupt context switch causes the interrupt vector to be fetched, the old WP, PC, and ST to be saved in the new workspace, and the new WP and PC to be loaded. Bits 12-15 of ST are loaded with a value of one less than the level of the interrupt being serviced. The old WP, PC, and ST are stored in the new workspace registers 13, 14, and 15. When the return instruction is executed, the old WP, PC, and ST are restored to the CPU. Since the ST contains the interrupt mask, the old interrupt level is also restored. Consequently, all interrupt service routines should terminate with the return instruction in order to restore the CPU to its state before the interrupt.

The linkage between two interrupt service routines is shown in *Figure 4-58* and the interrupt machine cycle sequence is shown in *Figure 4-59.*

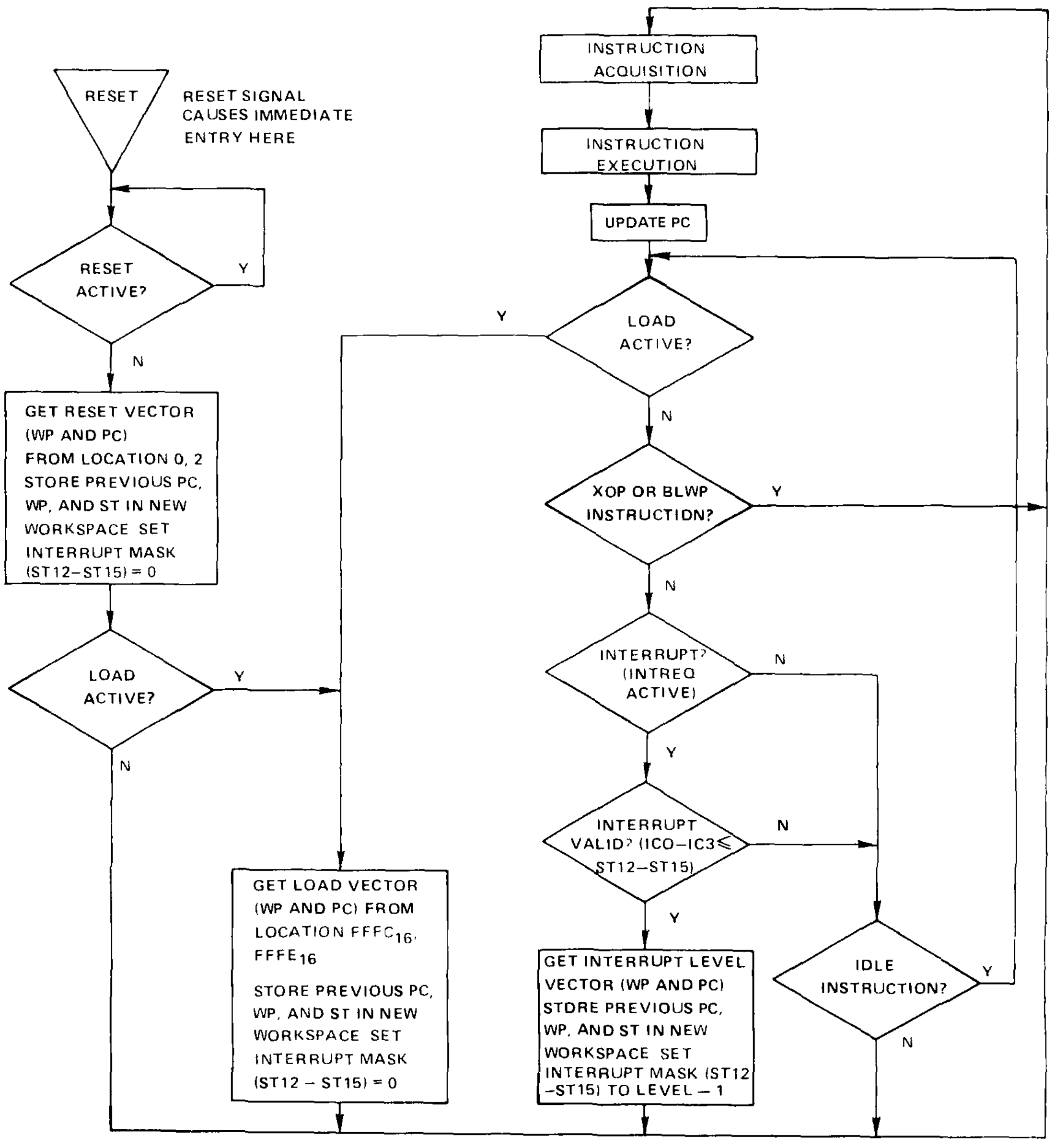

RESET
RESET SIGNAL CAUSES IMMEDIATE ENTRY HERE
RESET ACTIVE?
Y
N
GET RESET VECTOR (WP AND PC) FROM LOCATION 0, 2 STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE SET INTERRUPT MASK (ST12–ST15) = 0
LOAD ACTIVE?
Y
N
INSTRUCTION ACQUISITION
INSTRUCTION EXECUTION
UPDATE PC
LOAD ACTIVE?
Y
N
XOP OR BLWP INSTRUCTION?
Y
N
INTERRUPT? (INTREQ ACTIVE)
N
Y
INTERRUPT VALID? (IC0–IC3 ≤ ST12–ST15)
N
Y
GET LOAD VECTOR (WP AND PC) FROM LOCATION FFFC16, FFFE16
STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE SET INTERRUPT MASK (ST12 – ST15) = 0
GET INTERRUPT LEVEL VECTOR (WP AND PC) STORE PREVIOUS PC, WP, AND ST IN NEW WORKSPACE SET INTERRUPT MASK (ST12 –ST15) TO LEVEL – 1
IDLE INSTRUCTION?
Y
N

Figure 4-57. TMS 9900 CPU Flow Chart

Interrupt Signals

The TMS 9900 has five inputs that control maskable interrupts. The $\overline{\text{INTREQ}}$ signal is active (low) when a maskable interrupt is pending. If $\overline{\text{INTREQ}}$ is active at the end of the instruction cycle, the CPU compares the priority code on IC0 through IC3 to the interrupt mask (ST12-ST15). If the interrupt code of the pending interrupt is equal to or less than the current interrupt mask, the CPU executes a vectored interrupt; otherwise, the interrupt request is ignored. The interrupt priority codes are shown in *Table 4-3.* Note that the level-0 interrupt code should not be used for external interrupts since level 0 is reserved for RESET.

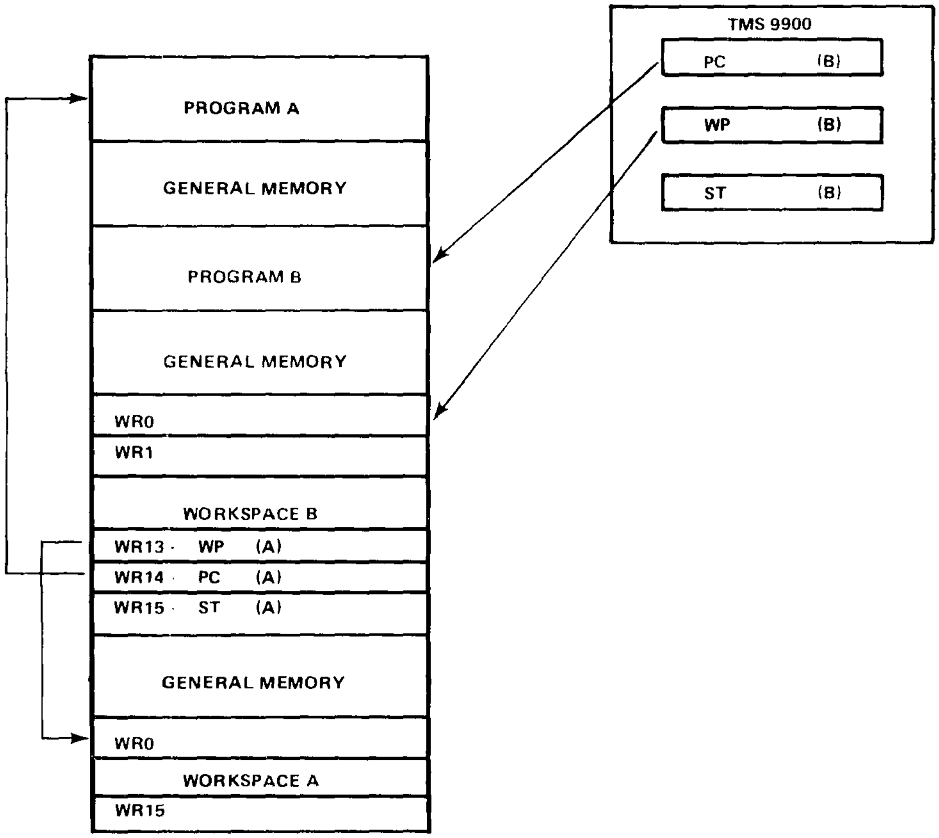


Figure 4-58. Interrupt Linkage

CYCLE	TYPE	FUNCTION
1	ALU	Set Up
2	Memory Read	Fetch New WP
3	ALU	Set Up
4	Memory Write	Store Status
5	ALU	Set Up
6	Memory Write	Store PC
7	ALU	Set Up
8	Memory Write	Store WP
9	ALU	Set Up
10	Memory Read	Fetch New PC
11	ALU	Set Up MAR for Next Instruction

Figure 4-59. Interrupt Processing Machine Cycle Sequence

Figure 4-60 illustrates the use of the TMS 9901 programmable system interface for generation of the interrupt code from individual interrupt input lines. The TMS 9901 provides six dedicated and nine programmable latched, synchronized, and prioritized interrupts, complete with individual enabling/disabling masks. Synchronization prevents transition of ICO-IC3 while the code is being read. A single-interrupt system with an arbitrarily chosen level-7 code is shown in *Figure 4-61.* The single-interrupt input does not need to be synchronized since the hardwired interrupt code is always stable.

Interrupt Masking

The TMS 9900 uses a four-bit field in the status register, ST12 through ST15, to determine the current interrupt priority level. The interrupt mask is automatically loaded with a value of one less than the level of the maskable interrupt being serviced. The interrupt mask is also affected by the load interrupt mask instruction (LIMI).

Since the interrupt mask is compared to the external interrupt code before an interrupt is recognized, an interrupt service routine will not be halted due to another interrupt of lower or equal priority unless a LIMI instruction is used to alter the interrupt mask. The LIMI instruction can be used to alter the interrupt-mask level in order to disable intervening interrupt levels. At the end of the service routine, a return (RTWP) restores the interrupt mask to its value before the current interrupt occurred.

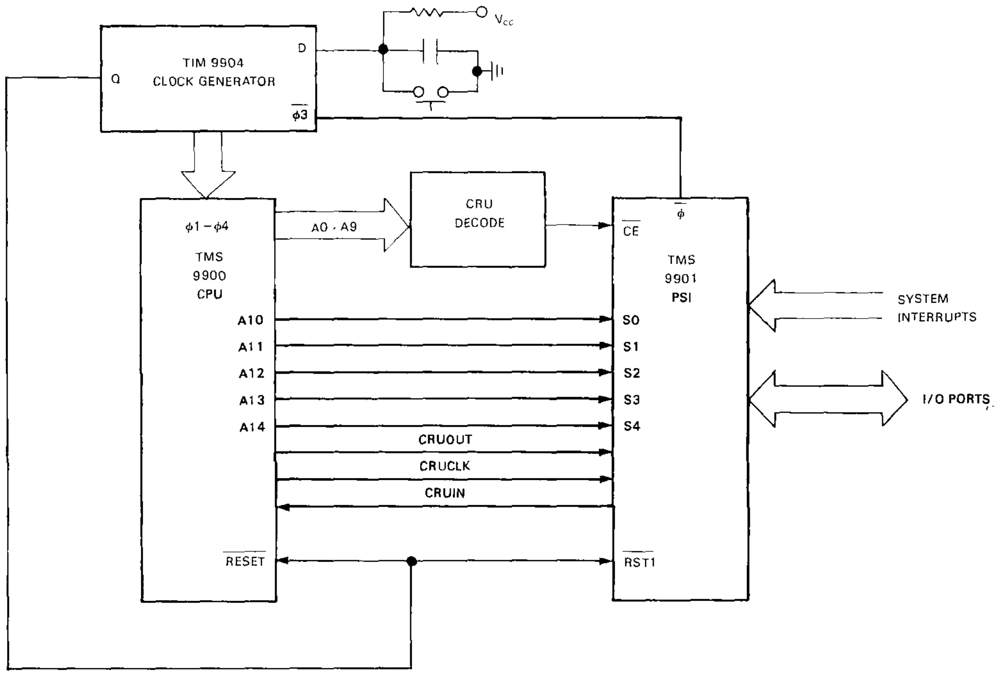


Figure 4-60. System With 15 External Interrupts

Table 4-3. Interrupt Priority Codes

Interrupt Level		Vector Location (Memory Address In Hex)	Device Assignment	Interrupt Mask Values To Enable Respective Interrupts (ST12 thru ST15)	Interrupt Codes IC0 thru IC3
(Highest priority)	0	00	Reset	0 through F*	0000
	1	04	External device	1 through F	0001
	2	08		2 through F	0010
	3	0C		3 through F	0011
	4	10		4 through F	0100
	5	14		5 through F	0101
	6	18		6 through F	0110
	7	1C		7 through F	0111
	8	20		8 through F	1000
	9	24		9 through F	1001
	10	28		A through F	1010
	11	2C		B through F	1011
	12	30		C through F	1100
	13	34		D through F	1101
	14	38	↓	E and F	1110
(Lowest priority)	15	3C	External device	F only	1111

*Level 0 can not be disabled.

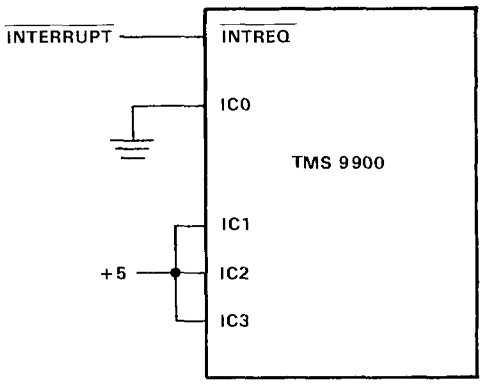


Figure 4-61. Single-Interrupt System

Note that the TMS 9900 actually generates the interrupt vector address using IC0-IC3 five clock cycles after it has sampled $\overline{\text{INTREQ}}$ and four clock cycles after it has compared the interrupt code to the interrupt mask in the status register. Thus, interrupt sources which have individual masking capability can cause erroneous operation if a command to the device to mask the interrupt occurs at a time when the interrupt is active and just after the TMS 9900 has sampled $\overline{\text{INTREQ}}$ but before the vector address has been generated using IC0-IC3.

The individual interrupt masking operation can be easily allowed if the masking instruction is placed in a short subroutine which masks all interrupts with a LIMI 0 instruction before individually masking the interrupt at the device, as shown in *Figure 4-62.*

```
INCORRECT

      XXX

      SBO     0                  SET MASK (INTERRUPT CAN OCCUR
                                 DURING SBO CAUSING ERRONEOUS
      YYY                        OPERATION)

CORRECT

      XXX

      BLWP    9                  (WR9)  =  ADDRESS OF SBW
                                 (WR10) =  ADDRESS OF SB1
      XXXX


SB1   LIMI    0                  CLEAR STATUS MASK TO INHIBIT INTERRUPTS
      MOV     @ 24 (13), 12      MOVE CRU BASE ADDRESS TO WR12
      SBO     0                  SET MASK
      RTWP                       RETURN


SBW   BSS     32                 SUBROUTINE WORKSPACE
```

Figure 4-62. External Interrupt Clearing Routine

Interrupt Processing Example

The routine in *Figure 4-63* illustrates the use of the LIMI instruction as a privileged or non-interruptable instruction. The level-5 routine sets a CRU bit and then loops until a corresponding CRU bit is true. The first instruction in the routine is completed before a higher priority interrupt can be recognized. The LIMI instruction, however, raises the CPU priority level to level 0 in order to disable all other maskable interrupts. Consequently, the level-5 routine will run to completion unless a RESET signal or a $\overline{\text{LOAD}}$ signal is generated. At the end of the routine, the RTWP instruction restores the CPU to its state before the level-5 interrupt occurred.

```
Level 5   LIMI   0      Disable Maskable INTREQs
          SBO    ACK    Set CRU Output Bit
Loop      TB     RDY    Test CRU Input Bit
          JNE    LOOP   Loop Until Input True
          RTWP          Return
```

Figure 4-63. LIMI Instruction Routine

ELECTRICAL REQUIREMENTS

Understanding the Electrical Specifications

A description of the interface to the 9900 would be incomplete without a set of specifications for the electrical signals which perform the functions described in the previous sections. Each pin of the 9900 may be characterized with a set of minimum and maximum voltage and current levels. In many cases, the switching characteristics, the rate of transition from the high state to the low state is also important. The detailed electrical specifications for each of the processors in the 9900 family are given in the *Product Data* chapter. A brief statement about the basic concepts of device characterization and data sheet specification is of value to designers with limited exposure to microprocessor and semiconductor memory products.

Specifications are given in two ways. First, absolute maximum ratings are given which simply define the limits of stress which the chip can withstand without damage. (*Figure 4-64* shows the absolute maximum ratings for the TMS 9900.) The normal design specification is the recommended operating conditions table *(Figure 4-65)* which specifies power supply limits, signal voltage levels, and the operating temperature range. In reading these two tables it is necessary to read the explanatory notes, one of which points out that the absolute maximum power supply voltages are specified with respect to the chip substrate or V_{BB} (pin 1). In the normal operating conditions, all voltages are specified with respect to the V_{SS} or ground (pins 26, 40). The four voltages given, V_{BB}, V_{CC}, V_{DD}, and V_{SS} are not actually four power supplies, but three power supplies: +5V, −5V, and +12V, with V_{SS} being the ground or reference point.

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

Supply voltage, V_{CC} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{DD} (see Note 1)	−0.3 to 20 V
Supply voltage, V_{SS} (see Note 1)	−0.3 to 20 V
All input voltages (see Note 1)	−0.3 to 20 V
Output voltage (with respect to V_{SS})	−2 V to 7 V
Continuous power dissipation	1.2 W
Operating free-air temperature range	0°C to 70°C
Storage temperature range	−55°C to 150°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: Under absolute maximum ratings voltage values are with respect to the most negative supply, V_{BB} (substrate), unless otherwise noted. Throughout the remainder of this section, voltage values are with respect to V_{SS}.

Figure 4-64. Absolute Maximum Ratings

RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB}	−5.25	−5	−4.75	V
Supply voltage, V_{CC}	4.75	5	5.25	V
Supply voltage, V_{DD}	11.4	12	12.6	V
Supply voltage, V_{SS}		0		V
High-level input voltage, V_{IH} (all inputs except clocks)	2.2	2.4	$V_{CC}+1$	V
High-level clock input voltage, $V_{IH(\phi)}$	$V_{DD}-2$		V_{DD}	V
Low-level input voltage, V_{IL} (all inputs except clocks)	−1	0.4	0.8	V
Low-level clock input voltage, $V_{IL(\phi)}$	−0.3	0.3	0.6	V
Operating free-air temperature, T_A	0		70	C

Figure 4-65. Recommended Operating Conditions

Input signals should be in the range from 2.2V to 6V (assuming V_{CC} is 5V) for the high level, the nominal design point being at 2.4V. Low level input voltage should be below 0.6V (but not less than −0.3V.) These specifications are not the same as the standard TTL specifications as far as the "worst case" design criteria are concerned. Care should be exercised when interfacing the 9900 with TTL circuits that loading of the TTL devices does not produce input voltages to the 9900 which are outside the specified range.

The clock signal voltages are substantially different from the TTL standard; however, the TMS 9904 is available to provide these signals.

The electrical characteristics specification, *Figure 4-66.* defines the current into or out of the 9900 chip at the operating voltage levels. The input current, I_I, is specified for four groups of input signals over a range of input voltages. For example, the input current for any input on the data bus (when reading data from the memory) is nominally ±50 microamps over the input voltage range from 0V to 5V (when V_{CC} is 5V). The current is negative (flowing out of the 9900) for low levels, and positive (into the 9900) for high levels. For "worst case" design the maximum values should be used.

Voltage specifications on the output pins show how the 9900 output devices drive external circuits. For the high level, V_{OH}, the voltage will be at least 2.4V but may go as high as 5V (V_{CC}) under the condition of output current of 0.4 mA. (Currents flowing out of the chip are shown as negative values.) When an output signal is at the low state, the output voltage, V_{OL}, will be no greater than 0.65V when the current flowing into the chip is 3.2 mA. Although the I-V characteristic of the output circuit is nonlinear, a second data point is given: if the current is 2 mA, the voltage will be no greater than 0.50V. These numbers tell the designer what the output drive circuit current sinking capability is. Two standard TTL loads (1.6 mA each) can be accommodated, but the V_{OL} level, as specified, may be as high as at 0.65V (the standard TTL specification for outputs is V_{OL} 0.4V.)

ELECTRICAL CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (UNLESS OTHERWISE NOTED)

PARAMETER			TEST CONDITIONS	MIN	TYP[†]	MAX	UNIT
I_I	Input current	Data bus during DBIN	$V_I = V_{SS}$ to V_{CC}		±50	±100	μA
		$\overline{WE}$, $\overline{MEMEN}$, DBIN, Address bus, Data bus during HOLDA	$V_I = V_{SS}$ to V_{CC}		±50	±100	
		Clock *	$V_I = -0.3$ to 12.6 V		±25	±75	
		Any other inputs	$V_I = V_{SS}$ to V_{CC}		±1	±10	
V_{OH}	High-level output voltage		$I_O = -0.4$ mA	2.4		V_{CC}	V
V_{OL}	Low-level output voltage		$I_O = 3.2$ mA			0.65	V
			$I_O = 2$ mA			0.50	
I_{BB}	Supply current from V_{BB}				0.1	1	mA
I_{CC}	Supply current from V_{CC}				50	75	mA
I_{DD}	Supply current from V_{DD}				25	45	mA
C_I	Input capacitance (any inputs except clock and data bus)		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		10	15	pF
$C_{i(\phi 1)}$	Clock-1 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi 2)}$	Clock-2 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		150	200	pF
$C_{i(\phi 3)}$	Clock-3 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi 4)}$	Clock-4 input capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
C_{DB}	Data bus capacitance		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		15	25	pF
C_O	Output capacitance (any output except data bus)		$V_{BB} = -5$, f = 1MHz, unmeasured pins at V_{SS}		10	15	pF

[†]All typical values are at $T_A = 25°C$ and nominal voltages
*D.C. Component of Operating Clock

Figure 4-66. Electrical Characteristics

The timing of the various signals on the TMS 9900 chip is shown in *Figure 4-67.* The fundamental propagation time from a clock phase pulse (leading edge) to the specified output is given as t_p and is typically 20 ns but is never more than 40 ns (worst case). The parameters t_{pLH} and t_{pHL} are the propagation delays from the appropriate clock signal to the low-to-high transition of the output (t_{pLH}) or the high-to-low transition of the output (t_{pHL}). For example, the WE signal makes its high-to-low transition 20 ns after ϕ1 clock, and makes a low-to-high transition 20 ns after the next ϕ1 clock. Most of the output signals make transitions 20 ns after the ϕ2 clock, and remain valid until the next 02 clock.

Additional information regarding design constraints based on the electrical specifications is given in the next section.

SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH} or t_{PHL} Propagation delay time, clocks to outputs	C_L = 200 pF		20		ns

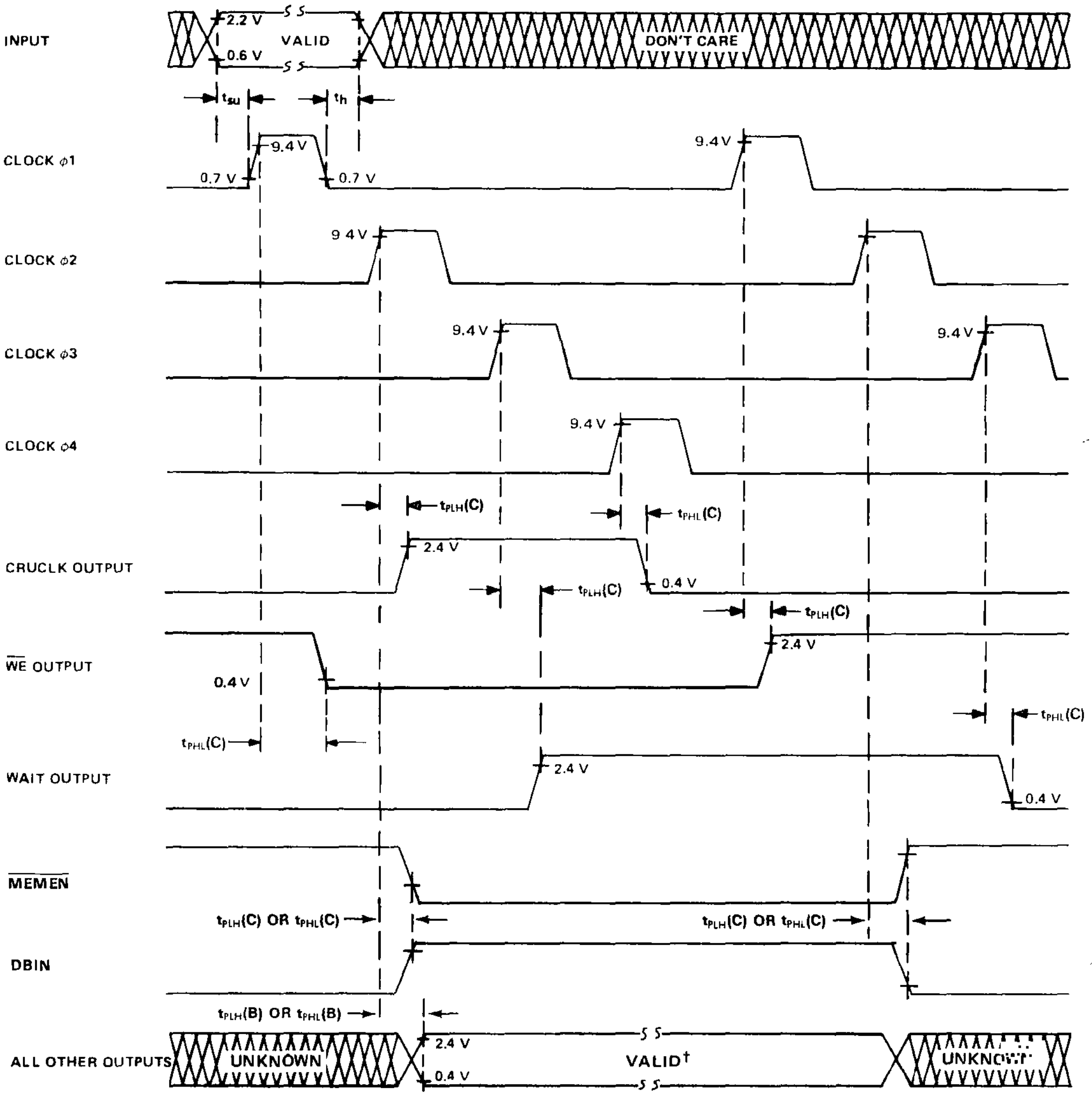


Figure 4-67. Switching Characteristics

Detailed Electrical Interface Specifications (TMS 9900)

This section reviews the TMS 9900 electrical requirements, including the system clock generation and interface signal characteristics. The "TMS9900 Data Manual" (Chapter 8) should be used for minimum and maximum values.

TMS 9900 Clock Generation

The TMS 9900 requires a non-overlapping four-phase clock system with high-level MOS drivers. Additional TTL outputs are typically required for external signal synchronization or for dynamic memory controllers. A single-chip clock driver, the TIM 9904, can be used to produce these clock signals. An alternative clock generator uses standard TTL logic circuits and discrete components.

The TMS 9900 requires four non-overlapping 12V clocks. The clock frequency can vary from 2 to 3 Megahertz. The clock rise and fall times must not exceed 100 nanoseconds and must be 10 to 15 nanoseconds for higher frequencies in order to satisfy clock pulse width requirements. While the clocks must not overlap, the delay time between clocks must not exceed 50 microseconds at lower frequencies. The typical clock timing for 3 MHz is illustrated in *Figure 4-68.*

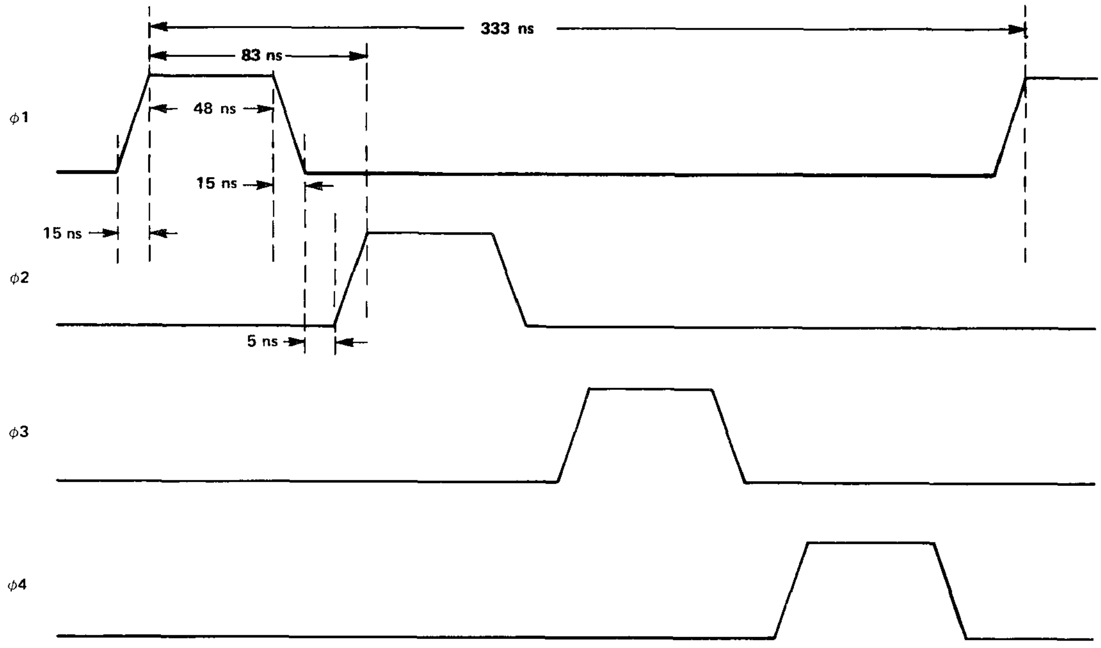


Figure 4-68. TMS 9900 Typical Clock Timing

TIM 9904 Clock Generator

The TIM 9904 (SN74LS362) is a single-chip clock generator and driver for use with the TMS 9900. The TIM 9904 contains a crystal-controlled oscillator, waveshaping circuitry, a synchronizing flip-flop, and quad MOS/TTL drivers as shown in *Figure 4-69.*

The clock frequency is selected by either an external crystal or by an external TTL-level oscillator input. Crystal operation requires a 16X input crystal frequency since the TIM 9904 divides the input frequency for waveshaping. For 3-megahertz operation, a 48-megahertz crystal is required. The LC tank inputs permit the use of overtone crystals. The LC network values are determined by the network resonant frequency:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

For less precise frequency control, a capacitor can be used instead of the crystal.

The external-oscillator input can be used instead of the crystal input. The oscillator input frequency is 4X the output frequency. A 12-megahertz input oscillator frequency is required for a 3-megahertz output frequency. A 4X TTL-compatible oscillator output (OSCOUT) is provided in order to permit the derivation of other system timing signals from the crystal or oscillator frequency source.

The oscillator frequency is divided by four to provide the proper frequency for each of the 4-clock phases. A high-level MOS output and an inverted TTL-compatible output is provided by each clock phase. The MOS-level clocks are used for the TMS 9900 CPU while the TTL clocks are used for system timing.

The D-type flip-flop is clocked by $\phi3$ and can be used to synchronize external signals such as a $\overline{\text{RESET}}$. The Schmitt-triggered input permits the use of an external RC network for power-on $\overline{\text{RESET}}$ generation. The RC values are dependent on the power supply rise time and should hold $\overline{\text{RESET}}$ low for at least three clock cycles after the supply voltages reach the minimum voltages.

All TIM 9904 TTL-compatible outputs have standard short circuit protection. The high-level MOS clock outputs, however, do not have short circuit protection.

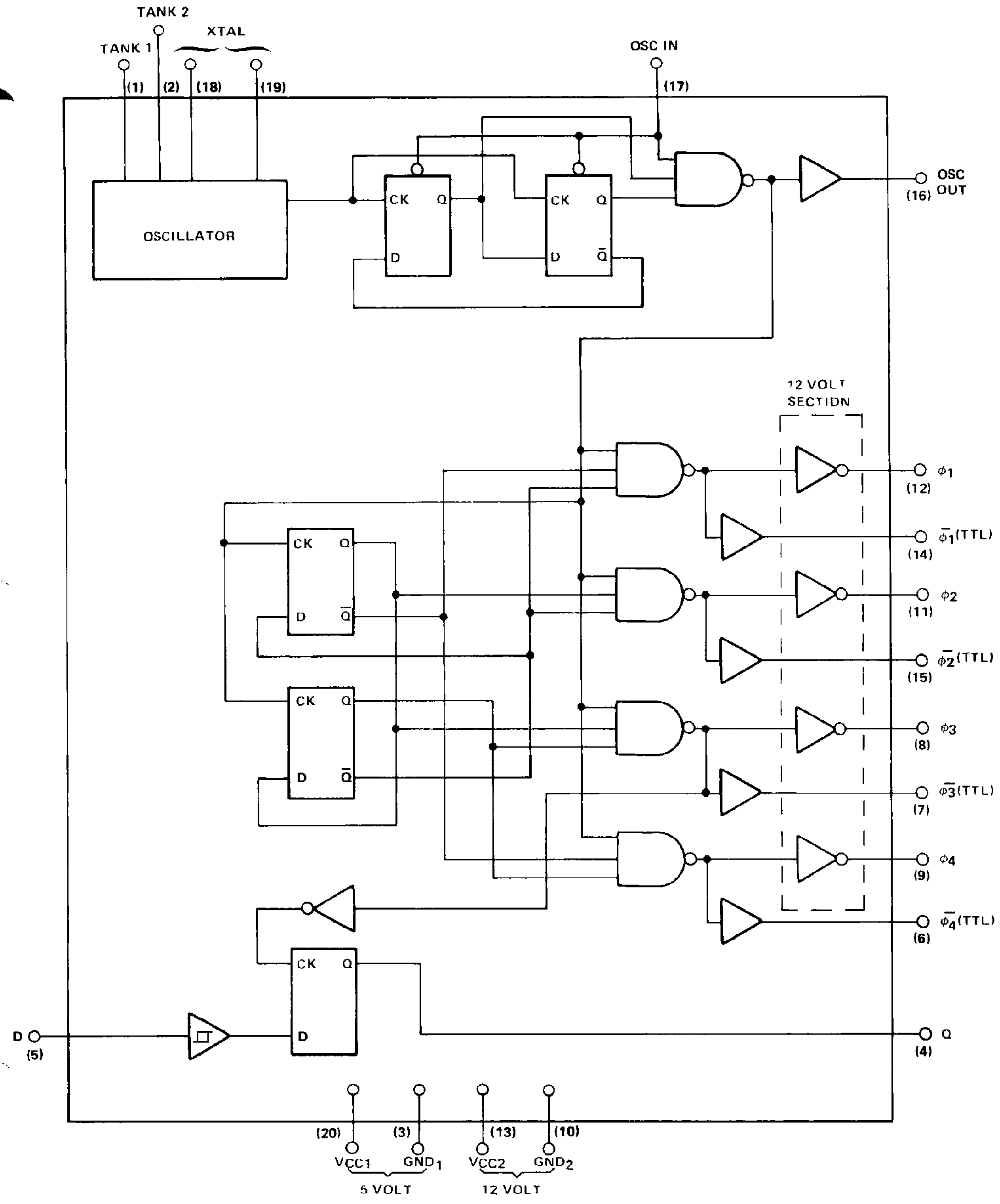


Figure 4-69. TIM 9904 Clock Generator

This driver uses inexpensive 2N3703s and 2N3704s and broad tolerance passive components. Resistor tolerances can be 10% with capacitor variations as much as 20% without affecting its performance noticeably. It shows very little sensitivity to transistor variations and its propagation times are largely unaffected by output capacitive loading. It produces rise times in the 10-12 ns region with fall times from 8-10 ns, driving 200 pF capacitive loads. Propagation times for this driver are such that it produces an output pulse that is wider than its input pulse. This driver can easily be used at 3 megahertz without special selection of components. It does have the disadvantage of taking nine discrete components per driver, but if assembly costs are prohibitive, these can be reduced by using two Q2T2222 and two Q2T2905 transistor packs. The Q2T2222 is basically four NPN transistors of the 2N2222 type while the Q2T2905 has four PNP, 2N2905 type transistors in single 14-pin dual-in-line packages. Thus, all four drivers can be built using two packages each of these quad packs.

TMS 9900 Signal Interfacing

The non-clock CPU inputs and outputs are TTL compatible and can be used with bipolar circuits without external pull-up resistors or level shifters. The TMS 9900 inputs are high impendance to minimize loading on peripheral circuits. The TMS 9900 outputs can drive approximately two TTL loads, thus eliminating the need for buffer circuits in many systems.

Switching Levels

The TMS 9900 input switch levels are compatible with most MOS and TTL circuits and do not require pull-up resistors to reach the required high-level input switching voltage. The TMS 9900 output levels can drive most MOS and bipolar inputs. Some typical switching levels are shown in *Table 4-4.*

Table 4-4. *Switch Levels*

SWITCHING LEVEL (V)	TMS 9900	TMS 2708	TMS 4042-2	SN 74XX	SN 74LSXX
V_{IH} min	2.2	3.0	2.2	2.0	2.0
V_{IL} max	0.6	0.65	0.65	0.8	0.7
V_{OH}* min	2.4	3.7	2.2	2.4	2.7
V_{OL} max	0.5	0.45	0.45	0.5	0.5

*V_{OH} exceeds 2.4 V as shown in Figure 4-70.

It should be noted that some MOS circuits such as the TMS 4700 ROM and the TMS 2708 EPROM have a minimum high-level input voltage of 3 V to 3.3 V, which exceeds the TMS 9900 minimum high-level output voltage of 2.4 V. The TMS 9900 high-level output voltage exceeds 3.3 V; however, longer transition times as shown in *Figure 4-70* are required.

Loading

The TMS 9900 has high-impedance inputs to minimize loading on the system buses. The CPU data bus presents a *maximum* current load of ± 100 μA when DBIN is high. $\overline{\text{WE}}$, $\overline{\text{MEMEN}}$, and DBIN cause a *maximum* current load of ± 100 μA during HOLDA. Otherwise, the TMS 9900 inputs present a current load of only ± 10 μA. The data bus inputs have a 25-picofarad input capacitance, and all other non-clock inputs have a 15-picofarad input capacitance.

The TMS 9900 outputs can drive approximately two standard TTL loads. Since most memory devices have high-impedance inputs, the CPU can drive small memory systems without address or data buffers. If the bus load exceeds the equivalent of two TTL unit loads, external buffers are required.

The TMS 9900 output switching characteristics are determined for approximately 200 picofarads. Higher capacitive loads can be driven with degraded switching characteristics as shown in *Figure 4-71.*

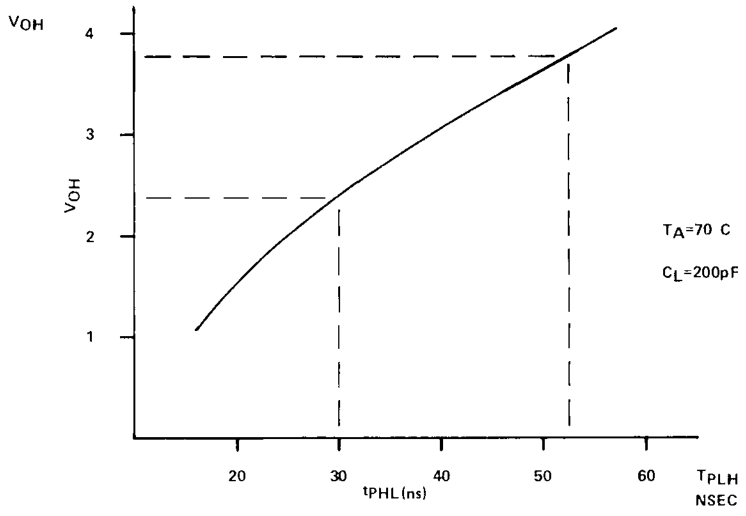


Figure 4-70. t_{PLH} *vs* V_{OH} *Typical Output Levels*

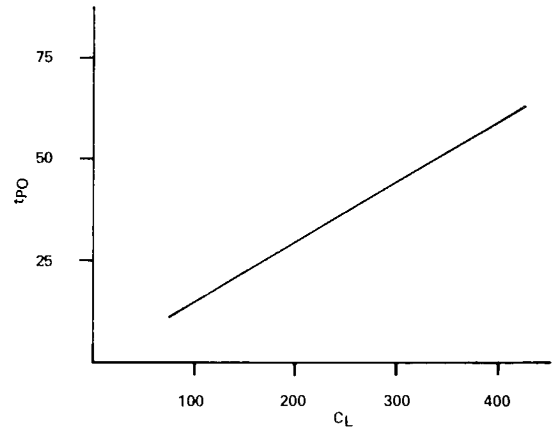


Figure 4-71. t_{PO} *vs Load Capacitance (Typical)*

Recommended Interface Logic

The TMS 9900 is compatible with the logic from any of the common TTL logic families. The Texas Instruments low-power Schottky logic circuits are, however, recommended for use in microprocessor systems. The SN74LSXX circuits have higher impedance inputs than standard TTL, allowing more circuits to be used without buffering. The SN74LSXX gates also consume less power at similar switching speeds. Texas Instruments has a wide assortment of bipolar support circuits which can be used with the TMS 9900, as shown in *Table 4-5.* Note that five circuits which are particularly useful in many applications have been dual symbolized with TIM 99XX numbers for easy reference.

There are a number of buffer circuits available for use in TMS 9900 systems. The SN74S241 and SN74LS241 non-inverting octal buffers with three-state outputs can be used as memory address drivers or as bidirectional data transceivers. The SN74S240 and SN74LS240 are similar, but with inverted outputs. The SN74LS241 can be used as either a memory-address buffer or as a transceiver for bidirectional data transfers. The use of a single circuit type for both functions can result in a lower inventory and parts cost. The buffer switching times can be derated for higher capacitive loading as required.

System Layout

The pin assignments of the TMS 9900 are such that sets of signals (data bus, address bus, interrupt port, etc.) are grouped together. The layout of a printed circuit board can be simplified by taking advantage of these groups by locating associated circuitry (address buffers, interrupt processing hardware, etc.) as close as possible to the TMS 9900 interface. Shortened conductor runs result in minimal noise and compact and efficient utilization of printed circuit board area.

It is particularly important that the drivers for $\phi 1$—$\phi 4$ be located as close as possible to the inputs of the TMS 9900, since these signals have fast rise and fall times while driving fairly high capacitance over a wide voltage range. The 12 volt supply to the clock drivers should be decoupled with both high (15μF) and low (0.05μF) value capacitors in order to filter out high and lower frequency variations in supply voltage.

Table 4-5. TMS 9900 Bipolar Support Circuits

DEVICE	FUNCTION	PACKAGE
	BUFFERS (3-STATE)	
SN74125	QUAD Inverting Buffer	14
SN74126	QUAD Inverting Buffer	14
SN74LS240	OCTAL Inverting Buffer/Transceiver	20
SN74LS241	OCTAL Noninverting Buffer/Transceiver	20
SN74LS242	OCTAL Inverting Transceiver	14
SN74LS243	OCTAL Noninverting Transceiver	14
SN74S240	OCTAL Inverting Buffer/Transceiver	20
SN74S241	OCTAL Noninverting Buffer/Transceiver	20
SN74365	Hex Noninverting Buffer	16
SN74366	Hex Inverting Buffer	16
SN74367	Hex Noninverting Buffer	16
SN74368	Hex Inverting Buffer	16
	LATCHES	
SN74LS259 (TIM9906)	OCTAL Addressable Latch	16
SN74LS373	OCTAL Transparent Latch (3-state)	20
SN74LS412	OCTAL I/O Port (3-state)	24
	DATA MULTIPLEXERS	
SN74LS151	OCTAL Multiplexer	16
SN74LS251 (TIM9905)	OCTAL Multiplexer (3-state)	16
	OTHER SUPPORT CIRCUITS	
SN74148 (TIM 9907)	Priority Encoder	16
SN74LS348 (TIM9908)	Priority Encoder	16
SN74LS74	Dual D-type flip-flop	14
SN74LS174	Hex D-type flip-flop	16
SN74LS175	Qual D-type flip-flop	16
SN74LS37	QUAD 2-Input nand Buffers	14
SN74LS362 (TIM9904)	Clock Generator	20

All voltage inputs to the TMS 9900 should be decoupled at the device. Particular attention should be paid to the +5 volt supply. All data and address lines are switched simultaneously. The worst-case condition occurs when all data and address signals switch to a low level simultaneously and they are each sinking 3.2 mA. It is thus possible for the supply current to vary nearly 100 mA over a 20 ns interval. Careful attention must be paid by the designer to avoid supply voltage spiking. The exact values for capacitors should be determined empirically, based on actual system layout and drive requirements.

TMS 9940 MICROCOMPUTER

The TMS9940 is a microcomputer chip in a 40-pin package which includes all of the elements of a computer, that is, memory, I/O and utilities in addition to ALU and control. Useful in a wide variety of dedicated control functions, it contains a 2k × 8 EPROM program memory and a 128 × 8 RAM for data, a 14 bit interval timer, and a multiprocessor system interface. Although the memory organization on chip is in 8 bit bytes, the instructions are the same 16-bit instructions of the 9900 family.

While most of the instructions are identical to the instruction set of the 9900, there are 68 instructions in the 9940 set (as opposed to 69 in the TMS9900) including three new ones. The differences in the instruction set are illustrated by the following list of instructions.

DCA	Decimal Correct for BCD add	Added Instructions
DCS	Decimal Correct for BCD subtract	
LIIM	Load Interrupt mask	
RSET	(external instructions in 9900)	Deleted Instructions
CKOF		
CKON		
LREX		
IDLE	Put processor into the idle state	Hardware in the 9940

The first three of the instructions in the above list are new instructions and are unique to the 9940 microcomputer. The DCA and DCS instructions perform decimal correct for BCD arithmetic. The LIIM instruction is a single word instruction to load the interrupt mask. (This instruction should be contrasted with the LIMI instruction of the 9900 set which performs the same function but occupies two memory words.) The idle instruction, an external instruction in the 9900 set, is now implemented in hardware. Four instructions in the list are not implemented in the 9940; they are external instructions in the 9900 set.

Pin Assignments and Functional Control

One of the most extraordinary features of the TMS9940 is the I/O structure in which 32 pins of the 40-pin package are software assignable. That is, they do not perform single, predefined, hard-wired functions, but instead are under the control of the programmer in structuring input/output functions. *Table 4-6* lists the functions of four specific bits in the CRU which are called configuration bits. Because these four bits are assigned specific locations in the CRU output field and are therefore addressable and accessible via CRU output instructions, the pins of the package may be dynamically reassigned during program execution.

Table 4-6. Configuration Bit Functions

Configuration Bit	*Function*
0	External CRU expansion
1	Multiprocessor
2	Clock output for sync
3	Power down

Table 4-7 describes the way these four configuration bits assign the individual general-purpose I/O pins to specific functions. In effect, each of the pins may serve two or three functions, as described in the table. *Table 4-8* defines the functions of addressable CRU locations. The first 256 locations, addresses 000 through 0FF, are for external expansion of the general I/O to an additional 245 (256 less 11 used for expansion). It is important to note here that these 256 bits are two fields of 256 bits each, one for input and one for output. Addresses 100 through 17F are not used, and address 180 through 1DF are used internally.

Notice in *Table 4-8* that CRU addresses 183, 184, 185 and 186 locate the four configuration bits. It is via the setting or resetting of these individual bits that the I/O configuration is established.

Four other significant features should be pointed out.
One: The interrupt structure includes four levels of interrupt as opposed to the 16-level interrupt capability of the general 9900 microprocessor group.
Two: There is an on-chip timer, or event counter.
Three: 32 bits of CRU I/O are implemented on the chip. (Addresses 1E0-1EFF)
Four: A multiprocessor system interface is constructed as part of the CRU I/O.

Interrupts

The four interrupt levels are shown in the table below.

Level 0	Reset	*Level 2*	Decrementer
Level 1	General Interrupt 1	*Level 3*	General Interrupt 2

Table 4-7. Configuration Bit Effects by Pin

	Configuration bit 0 (CRU Expansion)	
Pin	0	1
23	P0 (general I/O	A1
24	P1	A2
25	P2	A3
26	P3	A4
27	P4	A5
28	P5	A6
29	P6	A7
30	P7	A8
18	P8	CRUIN
17	P9	CRUOUT
16	P0	CRUCLK

	Configuration bit 1 (Multiprocessor)	
Pin	0	1
14	P11	$\overline{\text{TC}}$ (Clock)
11	P12	TD (Data)

	Configuration bit 2 (Sync)	
Pin	0	1
15	P13	ϕ (Clock)

	Configuration bit 3 (Power Down)	
Pin	0	1
10	P14	$\overline{\text{HLD}}$
9	P15	$\overline{\text{HLDA}}$
8	P16	$\overline{\text{IDLE}}$

▶4

Table 4-8. Functions of CRU Address

CRU Addresses	*Contents of R12*	*Input*	*Output*
000-0FF	000-1FE	CRU Expansion	CRU Expansion
100-17F	200-2FE	NA	NA
180	300	Test for Interrupt 1	
181	302	Test for Decrementer Interrupt	Clear Decrementer
182	304	Test for Interrupt 2	
183	306		Set Configuration Bit 0
184	308		Set Configuration Bit 1
185	30A		Set Configuration Bit 2
186	30C		Set Configuration Bit 3
187-18F	30E-31E	NA	NA
190-19D	320-33A	Read Decrementer Value	Load Decrementer Value
19E	33C		TE (Timer/Event Cntr)
19F	33E		
1A0-1AF	340-35E	Read MPSI Value	Load MPSI Value
1B0-1BF	360-37E	Read Flag Register	Set Flag Register
1C0-1DF	380-3BE		Set I/O Direction for P0-P31
1E0-1FF	3C0-3FE	P0-P31 Input Data	P0-P31 Output Data

The 9940 implements interrupts using the same context switch concept of the 9900. Thus, the interrupt vectors for the four interrupt levels must be stored in the first 16 words of the 9940's program memory. As is described in a subsequent paragraph, the decrementer acts like a counter in an external piece of hardware in that after the contents of the circuit have been decremented to zero an interrupt signals the processor to perform a context switch and perform whatever function was programmed as the service routine for the decrementer. The reset, INT 1, and INT 2 interrupt signals are available to external hardware.

Since there is no $\overline{\text{INTREQ}}$ (interrupt request) signal input for the 9940, an interrupt input must be set and remain set until acknowledged. In fact, the acknowledgement of an interrupt must include instructions to reset holding flip-flops (if used) via CRU operations.

In the 9940, the interrupt input may be masked (as in all 9900 processors) but there are specific CRU bits which, if tested, will reveal pending interrupts which are not being serviced. Thus, the programmer may wish to mask interrupts but still be aware (via TB instructions to CRU locations 180, 181 and 182 as shown in *Table 4-8*) of the interrupt input status.

Decrementer

A timer/event counter is implemented on the 9940 chip to introduce interrupts after a predefined time period or number of events. A set of dedicated CRU addresses define the location of decrementer input and output registers. A value may be loaded into the decrementer via an LDCR instruction which loads CRU locations $190_{16} - 19D_{16}$. Likewise, the current value of the decrementer may be read via an STCR instruction identifying the same CRU field.

When the decrementer contents count down to zero, an interrupt is issued. The context switch thus activated automatically clears the interrupt request.

As a timer, the decrementer counts down at the rate of 1/30 of the oscillator frequency. With a clock frequency of 5 MHz, the time interval for counting is six microseconds.

As an event counter, the decrementer is first loaded with a value and it then counts down (one bit for each positive transition on pin 7) until it reaches zero. An interrupt is then issued.

CRU Implementation

One of the most important features of the 9940 is the manner in which the CRU is used to perform pin assignments and functional control as well as input and output. The major impact is that the external devices and some of the internal devices are under direct control of the programmer via CRU instructions. The major emphasis (see *Table 4-8*) is as follows.

32 bits of input — on-chip multiplexer
32 bits of output — on-chip flip flops
32 bit register defining signal direction (in or out) for the assignable pins
16 bit flag register — may be written or read
14 bit "clock" register—for loading the decrementer
14 bit "read" register—for reading the decrementer
16 bit shift register for receiving instructions in a multiprocessor application, or used for sending 16-bit information over the MPSI data line to other processors
14 bit decrementer (used as a timer or counter)
256 bit CRU expansion (input and output)

Pin assignments may be explained by showing the basic application concept, that of using the 32 bits of internal CRU. Here the only decision is one of signal direction. It is possible to set the configuration once during initialization and never change it. But this limits the total number of I/O signals to 32. It is permissible to change the signal direction of each pin as needed, thus obtaining full utilization of the 32 inputs and 32 outputs. The pins themselves (labelled P0-P31 in *Table 4-9*) serve as a dynamically configurable bidirectional CRU port. Data is addressed in the CRU address field 1E0 to 1FF. Direction control is established by writing a logical one for output or zero for input to the appropriate address(es) in the CRU field 1C0-1DF. Reading the addresses assigned for output is permissible and allows the program to interrogate or determine the status of the on-chip CRU output flip flops.

Functional assignments of the first 18 I/O signals may be accomplished as a "configuring" of the pins. As shown in *Table 4-9,* eighteen additional signals may pass through the pins corresponding to P0-P17. By setting configuration bit 0 for example, signals P0-P10 are no longer available to external hardware. Instead, the CRU expansion signals, A1-A8 and CRU controls, are available. Configuring may be accomplished by the following code.

LI R12,>200	Set CRU hardware base address at 100_{16}
SBO >83	Add 83_{16} to set CRU bit 183_{16}

(The LI instruction must set R12 to two times the hardware base address because the LSB is ignored. See the CRU explanation in the TMS9940 section of Chapter 8.)

Multiprocessor System Interface (MPSI)

A two-wire communication technique is provided so that the 9940 may exchange 16-bit data and/or instructions with other CPU's in a multiprocessor application. This capability allows the RAM to be used as an instruction memory for short subprograms downloaded from another processor. Since the technique is based on the CRU concept, the 9940 will easily interface with the processors in the 9900 family. In order to use this feature, configuration bit 1 must first be set via

```
LI R12, >200
SBO >84
```

Then the information flows in from an external processor and is clocked by the external processor so that this operation is completely transparent to the CPU. The sender must interrupt the receiver to cause reading of the input word via

LI R12, > 340	Address the MPSI register
STCR @BUFF, 0	Store 16 bits in memory location BUFF

Refer to *Table 4-8* for CRU addresses of this and other functions.

To send data out over the MPSI the 9940 must first have configuration bit 1 set, and then it simply executes

LDCR @BUFF, 0

to send out 16 bits of data from memory location BUFF. The switch into and out of "send" status is automatic.

Table 4-9. TMS 9940 Configurable Pins

Pin Number	*General I/O*	*CRU Address Data I/O*	*CRU Address for Direction Control*	*Alternate Function*	*Configuration Bit*	*CRU Address of Config. Bit*
23	P0	1E0	1C0	A1	0	183
24	P1	1E1	1C1	A2	0	183
25	P2	1E2	1C2	A3	0	183
26	P3	1E3	1C3	A4	0	183
27	P4	1E4	1C4	A5	0	183
28	P5	1E5	1C5	A6	0	183
29	P6	1E6	1C6	A7	0	183
30	P7	1E7	1C7	A8	0	183
18	P8	1E8	1C8	CRUIN	0	183
17	P9	1E9	1C9	CRUOUT	0	183
16	P10	1EA	1CA	CRUCLK	0	183
14	P11	1EB	1CB	$\overline{\text{TC}}$	1	184
11	P12	1EC	1CC	TD	1	184
15	P13	1ED	1CD	$\overline{\phi}$	2	185
10	P14	1EE	1CE	$\overline{\text{HLD}}$	3	186
9	P15	1EF	1CF	$\overline{\text{HLDA}}$	3	186
8	P16	1F0	1D0	$\overline{\text{IDLE}}$	3	186
7	P17	1F1	1D1	EC	—	19E
6	P18	1F2	1D2			
5	P19	1F3	1D3			
4	P20	1F4	1D4			
3	P21	1F5	1D5			
2	P22	1F6	1D6			
1	P23	1F7	1D7			
31	P24	1F8	1D8			
32	P25	1F9	1D9			
33	P26	1FA	1DA			
34	P27	1FB	1DB			
35	P28	1FC	1DC			
36	P29	1FD	1DD			
38	P30	1FE	1DE			
39	P31	1FF	1DF			

Summary

The 9940 is a powerful member of the 9900 family with execution techniques which are actually faster than the TMS9900. In fact, because of its higher speed clock (5 MHz) and a fast on-chip execution microcycle for register location, the average throughput is 20% faster than the standard 9900 devices. The assignability of the package pins via software adds a new dimension to microprocessor technology for improved flexibility and performance.

For detailed information on this part, see the 9940 section of Chapter 8.

COMPLETE LISTING OF MACHINE CYCLES

In order to complete the description of instruction execution, the individual instruction execution cycles are given in this section. Each machine cycle consists of two or more clock cycles (depending upon addressing mode) as defined herein. (*Note:* These machine cycles apply equally to the TMS 9980A/81 microprocessor, with the exception of the memory cycle as detailed below.) The 9900 family machine cycles are divided into three categories described in the following paragraphs.

Machine Cycles

ALU Cycle

The ALU cycle performs an internal operation of the microprocessor. The memory interface control signals and CRU interface control signals are not affected by the execution of an ALU cycle, which takes two clock cycles to execute.

Memory Cycle

The memory cycle primarily performs a data transfer between the microprocessor and the external memory device. Appropriate memory bus control signals are generated by the microprocessor as a result of a memory cycle execution. The memory cycle takes 2 + W (where W is the number of wait states) clock cycles to execute.

In the TMS 9980A/81, which has an 8-bit data bus, the memory cycle is composed of two data transfers to move a complete 16-bit word. The TMS 9980A/81 memory cycle takes 4 + 2W (where W is the number of wait states) clock cycles to execute. For the TMS 9980A/81 the following machine cycle sequences replace the memory sequences used in the instruction discussion.

CYCLE				
1	Memory read/write	AB	=	Address of most significant byte (A13 = 0)
		DB	=	Most significant byte
2	Memory read/write	AB	=	Address of least significant byte (A13 = 1)
		DB	=	Least significant byte

CRU Cycle

The CRU cycle performs a bit transfer between the microprocessor and I/O devices. It takes two clock cycles to execute. The address of the CRU bit is set up during the first clock cycle. For an input operation the CRUIN line is sampled by the microprocessor during the second clock cycle. For an output operation the data bit is set up on the CRUOUT line at the same time the address is set up. The CRUCLK line is pulsed during the second clock cycle of the CRU output cycle. Please refer to the specific 99XX microprocessor data manual for timing diagrams.

The 9900 executes its operations under the control of a microprogrammed control ROM. Each microinstruction specifies a machine cycle. A microprogram specifies a sequence of machine cycles. The 9900 executes a specific sequence of machine cycles for a specific operation. These sequences are detailed on the following pages. The information can be used by the systems designers to determine the bus contents and other interface behavior at various instants during a certain 9900 operation. This description is maintained at the address bus (AD) and data bus (DB) levels.

9900 Machine Cycle Sequences

Most 9900 instructions execution consists of two parts: 1) the data derivation and 2) operation execution. The data derivation sequence depends on the addressing mode for the data. Since the addressing modes are common to all instructions, the data derivation sequence is the same for the same addressing mode, regardless of the instruction. Therefore, the data derivation sequences are described first. These are then referred to in appropriate sequence in the instruction execution description.

Terms and Definitions

The following terms are used in describing the instructions of the 9900:

TERM	DEFINITION
B	Byte Indicator (1 = byte, 0 = word)
C	Bit count
D	Destination address register
DA	Destination address
IOP	Immediate operand
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
SD	Source data register internal to the TMS 9900 microprocessor*
W	Workspace register
SRn	Workspace register n
(n)	Contents of n
Ns	Number of machine cycles to derive source operand
Nd	Number of machine cycles to derive destination operand
AB	Address Bus of the TMS 9900
DB	Data Bus of the TMS 9900
NC	No change from previous cycle

**Note:* The contents of the SD register remain latched at the last value written by the processor unless changed by the ALU. Therefore, during all memory read or ALU machine cycles the SD register and hence the data bus will contain the operand last written to the data bus by the CPU or the results of the last ALU cycle to have loaded the SD register.

Data Derivation Sequence

Workspace Register

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address
		DB = Operand

Workspace Register Indirect

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address
		DB = Workspace register contents
2	ALU	AB = NC
		DB = SD
3	Memory read	AB = Workspace register content
		DB = Operand

Workspace Register Indirect Auto-Increment (Byte-Operand)

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address
		DB = Workspace register contents
2	ALU	AB = NC
		DB = SD
3	Memory write	AB = Workspace register address
		DB = (WRn) + 1
4	Memory read	AB = Workspace register contents
		DB = Operand

Workspace Register Indirect Auto-Increment (Word Operand)

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = Workspace register address
		DB = Workspace register contents
2	ALU	AB = NC
		DB = SD
3	ALU	AB = NC
		DB = SD
4	Memory write	AB = Workspace register address
		DB = (WRn) + 2
5	Memory read	AB = Workspace register contents
		DB = Operand

Symbolic

CYCLE	TYPE	DESCRIPTION

1	ALU	AB = NC
		DB = SD

2	ALU	AB = NC
		DB = SD

3	Memory read	AB = PC + 2
		DB = Symbolic address

4	ALU	AB = NC
		DB = 0000_{16}

5	Memory read	AB = Symbolic address
		DB = Operand

Indexed

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = Workspace register address
		DB = Workspace register contents

2	ALU	AB = NC
		DB = SD

3	Memory read	AB = PC + 2
		DB = Symbolic address

4	ALU	AB = PC + 2
		DB = Workspace register contents

5	Memory read	AB = Symbolic address + (WRn)
		DB = Operand

INSTRUCTION EXECUTION SEQUENCE

A, AB, C, CB, S, SB, SOC, SOCB, SZC, SZCB, MOV, MOVB, COC, CZC, XOR

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate sequence for source data addressing mode, from the data derivation sequences	(Note 1)

3 + Ns	ALU	AB = NC
		DB = SD

Nd	Insert appropriate sequence for destination data addressing mode from the data derivation sequences	(Note 2, 3)

4 + Ns + Nd	ALU	AB = NC
		DB = SD

5 + Ns + Nd	Memory write	AB = DA (Note 4)
		DB = Result

NOTES:

1) Since the memory operations of the 9900 microprocessor family fetch or store 16-bit words, the source and the destination data fetched for byte operations are 16-bit words. The ALU operates on the specified bytes of these words and modifies the appropriate byte in the destination word. The adjacent byte in the destination word remains unaltered. At the completion of the instruction, the destination word, consisting of the modified byte and the adjacent unmodified byte, is stored in a single-memory write operation.
2) For MOVB instruction the destination data word (16 bits) is fetched. The specified byte in the destination word is replaced with the specified byte of the source-data word. The resultant destination word is then stored at the destination address.
3) For MOV instruction the destination data word (16 bits) is fetched although not used.
4) For C, CB, COC, CZC instructions cycle $5 + N_s + N_d$ above is an ALU cycle with AB = DA and DB = SD.

MPY (Multiply)

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data (multiplier) addressing mode	

3 + Ns	ALU	AB = NC
		DB = SD

4 + Ns	Memory read	AB = Workspace register address
		DB = Workspace register contents

5 + Ns	ALU	AB = NC
		DB = SD

6 + Ns	ALU	AB = NC
		DB = Multiplier

7 + Ns		Multiply the two operands

	16 ALU	AB = NC
		DB = MSH of partial product

24 + Ns	Memory write	AB = Workspace register address
		DB = MSH of the product

25 + Ns	ALU	AB = DA + 2
		DB = MSH of product

26 + Ns	Memory write	AB = DA + 2
		DB = LSH of the product

DIV (Divide)

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data (divisor) addressing mode	

3 + Ns	ALU	AB = NC
		DB = SD

4 + Ns	Memory read	AB = Address of workspace register DB = Contents of workspace register

5 + Ns	ALU	(Check overflow)
		AB = NC
		DB = Divisor

6 + Ns	ALU	(Skip if overflow to next instruction fetch)
		AB = NC
		DB = SD

7 + Ns	Memory read	AB = DA + 2
		DB = Contents of DA + 2

8 + Ns	ALU	AB = NC
		DB = SD

9 + Ns	ALU	AB = NC
		DB = SD

	Divide sequence consisting of Ni cycles where $48 \leq Ni \leq 32$. Ni is data dependent	AB = NC
		DB = SD

10 + Ns + Ni	ALU	AB = NC
		DB = SD

11 + Ns + Ni	Memory write	AB = Workspace register address
		DB = Quotient

12 + Ns + Ni	ALU	AB = DA + 2
		DB = Quotient

13 + Ns + Ni	Memory write	AB = DA + 2
		DB = Remainder

▶4

XOP

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	Instruction decode AB = NC DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3 + Ns	ALU	AB = NC
		DB = SD

4 + Ns	ALU	AB = NC
		DB = SA

5 + Ns	ALU	AB = NC
		DB = SD

6 + Ns	Memory read	AB = $40_{16} + 4 \times D$
		DB = New workspace pointer

CYCLE	TYPE	DESCRIPTION

7 + Ns	ALU	AB = NC
		DB = SA

8 + Ns	Memory write	AB = Address of WR11
		DB = SA

9 + Ns	ALU	AB = Address of WR15
		DB = SA

10 + Ns	Memory write	AB = Address of workspace register 15 DB = Status register contents

11 + Ns	ALU	AB = NC
		DB = PC + 2

12 + Ns	Memory write	AB = Address of workspace register 14 DB = PC + 2

13 + Ns	ALU	AB = Address of WR13
		DB = SD

14 + Ns	Memory write	AB = Address of workspace register 13 DB = WP

15 + Ns	ALU	AB = NC
		DB = SD

16 + Ns	Memory read	AB = $42_{16} + 4 \times D$
		DB = New PC

17 + Ns	ALU	AB = NC
		DB = SD

CLR, SETO, INV, NEG, INC, INCT, DEC, DECT, SWPB

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3 + Ns	ALU	AB = NC
		DB = SD

4 + Ns	Memory write	AB = Source data address
		DB = Modified source data

Note: The operand is fetched for CLR and SETO although not used.

ABS

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3 + Ns	ALU	Test source data
		AB = NC
		DB = SD

4 + Ns	ALU	Jump to 5′ + Ns if data positive
		AB = NC
		DB = SD

CYCLE	TYPE	DESCRIPTION

5+ns	ALU	Negate source
		AB = NC
		DB = SD

6+Ns	Memory write	AB = Source data address
		DB = Modified source data

5′+Ns	ALU	AB = NC
		DB = SD

X

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert the appropriate data derivation sequence according to the source data addressing mode	

3+Ns	ALU	AB = NC
		DB = SD

▶4

Note: Add sequence for the instruction specified by the operand.

B

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3+Ns	ALU	AB = NC
		DB = SD

Note: The source data is fetched, although it is not used.

BL

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3+Ns	ALU	AB = NC
		DB = SD

4+Ns	ALU	AB = Address of WR11
		DB = SD

5+Ns	Memory write	AB = Address of WR11
		DB = PC+2

Note: The source data is fetched although it is not used.

BLWP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
Ns	Insert appropriate data derivation sequence according to the source data addressing mode	
3+Ns	ALU	AB = NC
		DB = SD
4+Ns	ALU	AB = Address of WR15
		DB = NC
5+Ns	Memory write	AB = Address of workspace register 15
		DB = Status register contents
6+Ns	ALU	AB = NC
		DB = PC+2
7+Ns	Memory write	AB = Address of workspace register 14
		DB = PC+2
8+Ns	ALU	AB = Address or workspace register 13
		DB = SD
9+Ns	Memory write	AB = Address of workspace register 13
		DB = WP
10+Ns	ALU	AB = NC
		DB = SD
11+Ns	Memory read	AB = Address of new PC
		DB = New PC
12+Ns	ALU	AB = NC
		DB = SD

LDCR

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
Ns	Insert appropriate data derivation sequence	
3+Ns	ALU	AB = NC
		DB = SD
4+Ns	ALU	AB = NC
		DB = SD
5+Ns	ALU	AB = Address of WR12
		DB = SD
6+Ns	ALU	AB = Address of WR12
		DB = SD
7+Ns	Memory read	AB = Address of WR12
		DB = Contents of WR12
8+Ns	ALU	AB = NC
		DB = SD
C	Shift next bit onto CRUOUT line. Enable CRUCLK. Increment CRU bit address on AB. Iterate this sequence C times, where C is number of bits to be transferred.	AB = Address +2 Increments C Times
		DB = SD
9+Ns+C	ALU	AB = NC
		DB = SD

STCR

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

Ns	Insert appropriate data derivation sequence according to the source data addressing mode	

3+Ns	ALU	AB = NC
		DB = SD

4+Ns	Memory read	AB = Address of WR12
		DB = Contents of WR12

5+Ns	ALU	AB = NC
		DB = SD

6+Ns	ALU	AB = NC
		DB = SD

C	Input selected CRU bit. Increment CRU bit address. Iterate this sequence C times where C is the number of CRU bits to be input.	AB = Address + 2 C times
		DB = SD

7+Ns+C	ALU	AB = NC
		DB = SD

8+Ns+C	ALU	AB = NC
		DB = SD

C′	Right adjust (with zero fill) byte (if $C<8$) or word (if $8 < C < 16$).	AB = NC
		DB = SD

C′	$= 8\text{-}C\text{-}1$ if $C \leq 8$ $= 16\text{-}C$ if $8 < C \leq 16$	

9+Ns+C+C′	ALU	AB = NC
		DB = SD

10+Ns+C+C′	ALU	AB = NC
		DB = SD

11+Ns+C+C′	ALU	AB = Source address
		DB = SD

12+Ns+C+C′	Memory write	AB = Source address
		DB = I/O data

▶4

Note: For STCR instruction the 16-bit word at the source address is fetched. If the number of CRU bits to be transferred is ≤ 8, the CRU data is right justified (with zero fill) in the specified byte of the source word and source data word thus modified is then stored back in memory. If the bits to be transferred is > 8 then the source data fetched is not used. The CRU data in this case is right justified in 16-bit word which is then stored at the source address.

SBZ, SBO

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	AB = NC
		DB = SD

4	Memory read	AB = Address of WR12
		DB = Contents of WR12

5	ALU	AB = NC
		DB = SD

6	CRU	Set CRUOUT = 0 for SBZ = 1 for SBO AB = CRU Bit Address Enable CRUCLK

TB

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	AB = NC
		DB = SD

4	Memory read	AB = Address of WR12
		DB = Contents of WR12

5	ALU	AB = NC
		DB = SD

6	CRU	Set ST(2) = CRUIN
		AB = Address of CRU bit
		DB = SD

JEQ, JGT, JH, JHE, JL, JLE, JLT, JMP, JNC, JNE, JNO, JOC, JOP

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	Skip to cycle #5 if TMS 9900 status satisfies the specified jump condition
		AB = NC
		DB = SD

4	ALU	AB = NC
		DB = Displacement value

5	ALU	AB = NC
		DB = SD

SRA, SLA, SRL, SRC

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	Memory read	AB = Address of the workspace register
		DB = Contents of the workspace register
4	ALU	Skip to cycle #9 if C ≠ 0
		C = Shift count
		AB = NC
		DB = SD
5	ALU	AB = NC
		DB = SD
6	Memory read	AB = Address of WR0
		DB = Contents of WR0
7	ALU	AB = Source address
		DB = SD
8	ALU	AB = NC
		DB = SD
9		AB = NC
		DB = SD
C	Shift the contents of the specified workspace register in the specified direction by the specified number of bits. Set appropriate status bits.	
9+C	Memory write	AB = Address of the workspace register
		DB = Result
10+C	ALU	Increment PC
		AB = NC
		DB = SD

▶4

AI, ANDI, ORI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	ALU	AB = NC
		DB = SD
4	Memory read	AB = Address of workspace register
		DB = Contents of workspace register
5	Memory read	AB = PC+2
		DB = Immediate operand
6	ALU	AB = NC
		DB = SD
7	Memory write	AB = Address of workspace register
		DB = Result of instruction

CI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = NC
3	Memory read	AB = Address of workspace register
		DB = Contents of workspace register
4	ALU	AB = NC
		DB = SD
5	Memory read	AB = PC+2
		DB = Immediate operand
6	ALU	AB = NC
		DB = SD
7	ALU	AB = NC
		DB = SD

LI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	ALU	AB = NC
		DB = SD
4	Memory read	AB = PC+2
		DB = Immediate operand
5	ALU	AB = Address of workspace register
		DB = SD
6	Memory write	AB = Address of workspace register
		DB = Immediate operand

LWPI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	ALU	AB = NC
		DB = SD
4	Memory read	AB = PC+2
		DB = Immediate operand
5	ALU	AB = NC
		DB = SD

LIMI

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	ALU	AB = NC
		DB = SD
4	Memory read	AB = PC+2
		DB = Immediate data

CYCLE	TYPE	DESCRIPTION

5	ALU	AB = NC
		DB = SD

6	ALU	AB = NC
		DB = SD

7	ALU	AB = NC
		DB = SD

STWP, STST

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	AB = Address of workspace register DB = SD
4	Memory write	AB = Address of the workspace register DB = TMS 9900 internal register contents (WP or ST)

CKON, CKOF, LREX, RSET

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	AB = NC
		DB = SD

4	CRU	Enable CRUCLK
		AB = External instruction code
		DB = SD

5	ALU	AB = NC
		DB = SD

6	ALU	AB = NC
		DB = SD

IDLE

CYCLE	TYPE	DESCRIPTION

1	Memory read	AB = PC
		DB = Instruction

2	ALU	AB = NC
		DB = SD

3	ALU	AB = NC
		DB = SD

4	CRU	Enable CRUCLK
		AB = Idle code
		DB = SD

5	ALU	AB = NC
		DB = SD

6	ALU	AB = NC
		DB = NC

RTWP

CYCLE	TYPE	DESCRIPTION
1	Memory read	AB = PC
		DB = Instruction
2	ALU	AB = NC
		DB = SD
3	ALU	WP+30
4	Memory read	AB = Address of WR15
		DB = $\text{Status}_{\text{OLD}}$
5	Memory read	AB = Address of WR14
		DB = PC_{OLD}
6	Memory read	AB = Address of WR13
		DB = WP_{OLD}
7	ALU	AB = NC
		DB = SD

Machine-Cycle Sequence in Response to External Stimuli

RESET

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC
		DB = SD
2	ALU	AB = NC
		DB = SD
3	ALU	AB = 0
		DB = 0
4	Memory read	AB = 0
		DB = Workspace pointer
5	ALU	AB = NC
		DB = Status
6	Memory write	AB = Address of WR15
		DB = Contents of Status register
7	ALU	AB = NC
		DB = PC
8	Memory write	AB = Address of workspace register 14
		DB = PC+2
9	ALU	AB = Address of WR13
		DB = SD
10	Memory write	AB = Address of workspace register 13
		DB = WP
11	ALU	AB = NC
		DB = SD
12	Memory read	AB = 2
		DB = New PC
13	ALU	AB = NC
		DB = SD

*Occurs immediately after $\overline{\text{RESET}}$ is released following a minimum 3 cycle $\overline{\text{RESET}}$

LOAD

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC
		DB = SD
2	Memory read	AB = $FFFC_{16}$
		DB = Contents of $FFFC_{16}$
3	ALU	AB = NC
		DB = Status
4	Memory write	AB = Address of WR15
		DB = Contents of status register
5	ALU	AB = NC
		DB = PC
6	Memory write	AB = Address of WR14
		DB = PC + 2
7	ALU	AB = Address of WR13
		DB = SD
8	Memory write	AB = Address of workspace register 13
		DB = WP
9	ALU	AB = NC
		DB = SD
10	Memory read	AB = FFFE
		DB = New PC
11	ALU	AB = NC
		DB = SD

*Occurs immediately after last clock cycle of preceding instruction.

Interrupts

CYCLE	TYPE	DESCRIPTION
1*	ALU	AB = NC
		DB = SD
2	Memory read	AB = Address of interrupt vector
		DB = WP
3	ALU	AB = NC
		DB = Status
4	Memory write	AB = Address of WR15
		DB = Status
5	ALU	AB = NC
		DB = PC
6	Memory write	AB = Address of WR 14
		DB = PC+2
7	ALU	AB = Address of WR13
		DB = SD
8	Memory write	AB = Address of WR 13
		DB = WP
9	ALU	AB = NC
		DB = SD
10	Memory read	AB = Address of second word of interrupt vector
		DB = New PC
11	ALU	AB = NC
		DB = SD

*Occurs immediately after last clock cycle of preceding instruction

Timing

The timing of the ALU, CRU, and memory cycles is shown in *Figures 4-77, 78* and *79. Figure 4-80* shows the TMS9980A/81 memory cycle.

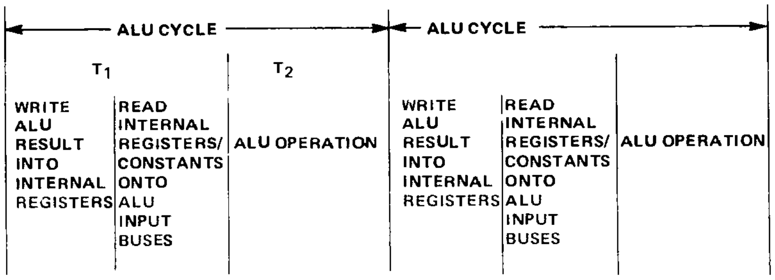

ALU CYCLE
ALU CYCLE
T1
T2
WRITE ALU RESULT INTO INTERNAL REGISTERS
READ INTERNAL REGISTERS/ CONSTANTS ONTO ALU INPUT BUSES
ALU OPERATION
WRITE ALU RESULT INTO INTERNAL REGISTERS
READ INTERNAL REGISTERS/ CONSTANTS ONTO ALU INPUT BUSES
ALU OPERATION

Figure 4-77. ALU Cycle

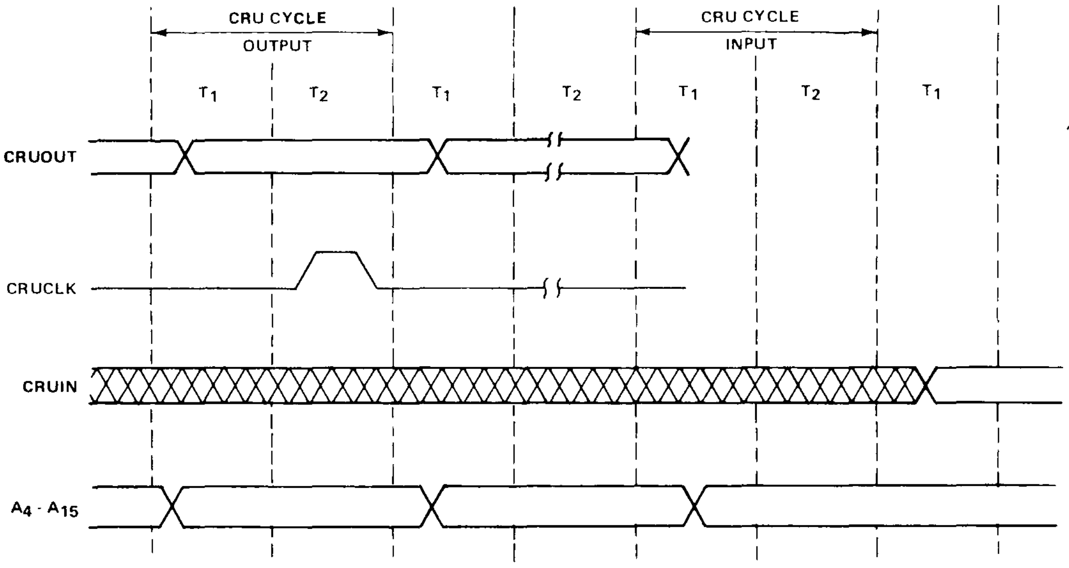


Figure 4-78. *CRU Cycle.*

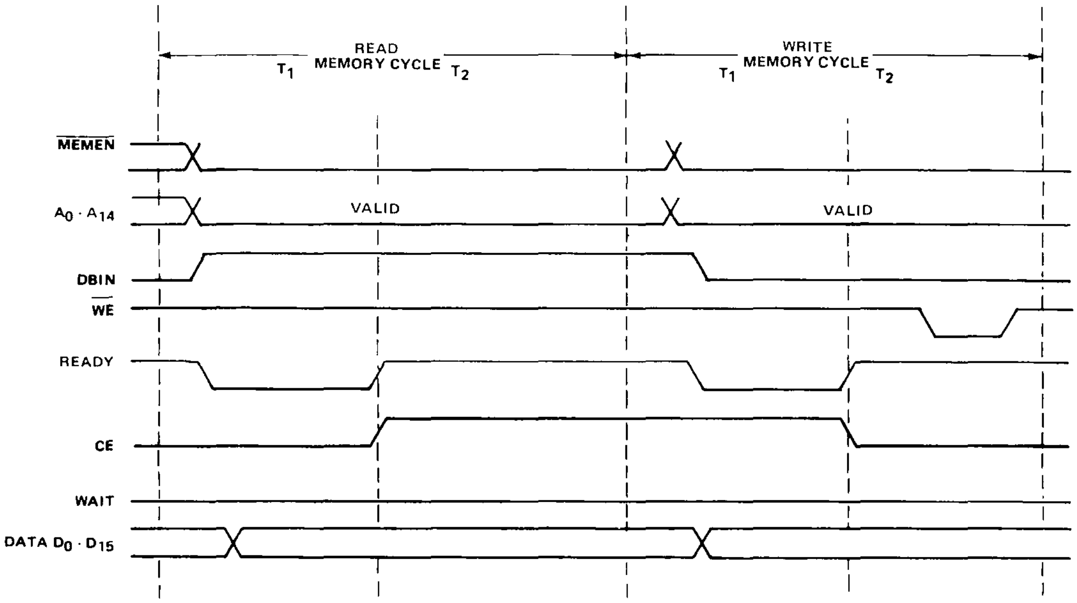


Figure 4-79. *TMS 9900 Memory Cycle (No Wait States)*

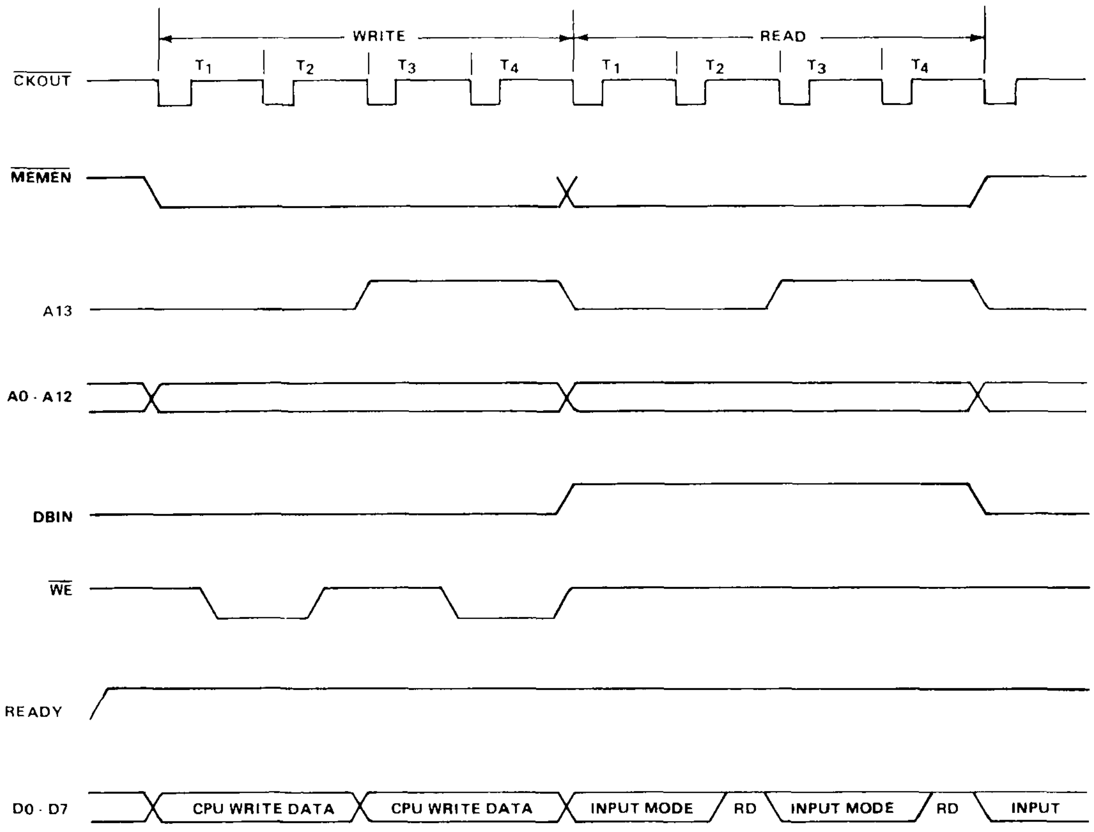


Figure 4-80. *TMS 9980A/81 Memory Cycle (No Wait States)*