Chapter 18
THE TEXAS INSTRUMENTS TMS 9900, TMS 9980, AND TMS 9940 PRODUCTS

The TMS 9900 was the first 16-bit microprocessor that could compete effectively in the minicomputer market. In fact, **the TMS 9900 is a one-chip implementation of the TM 990 series minicomputer Central Processing Units.**

The TMS 9900 is packaged as a 64-pin DIP; it generates signals for a 15-bit Address Bus and a separate 16-bit Data Bus, whereas other 16-bit microprocessors multiplex their Data and Address Busses. **The TMS 9980 series microprocessors are 40-pin DIP versions of the TMS 9900;** in order to reduce pin counts, the TMS 9980 series microprocessors access external memory via an 8-bit Data Bus and 14-bit Address Bus. **The TMS 9940 is a one-chip microcomputer** containing a subset of the TMS 9900 Central Processing Unit, together with on-chip memory and real-time clock logic.

The TMS 9900 product line has for some time been one of the enigmas of the microprocessor industry. Even a casual examination of the TMS 9900 instruction set shows that from the programmer's viewpoint, this microprocessor was at least two years ahead of its time. While it may have had problems competing in high-volume, simple applications, it was certainly the microprocessor of choice for data processing-type, program-intensive applications, yet it was not widely used in these markets.

The reason for this lack of acceptance has been poor support from Texas Instruments.

Texas Instruments initially offered little support for the TMS 9900 because this microprocessor was designed as a low-end product of the TM 990 minicomputer series. That is to say, customers were expected to develop products around the TM 990 minicomputers; then, if they chose to, they could build production models around the TMS 9900 microprocessor. This development path did not call for extensive TMS 9900 support. In all probability, Texas Instruments was caught by surprise by the buoyancy of the microprocessor market — as a market in its own right. Certainly, if Texas Instruments had given the TMS 9900 the same level of support that Intel gave the 8080A, we would see entirely different microprocessor product distributions today. But the TMS 9900 and its derivative products are powerful enough that the belated support they are now receiving from Texas Instruments will give the product line a reasonable share of future markets.

Texas Instruments now provides full support for the TMS 9900 microprocessor line.

TMS 9900 support devices are designed specifically for the TMS 9900; therefore, they are described in this chapter rather than in Volume 3. Support devices can be used with the TMS 9900, TMS 9980, or TMS 9940 products. The following devices are described:

The TIM 9904 Clock Generator
The TMS 9901 Programmable System Interface

Texas Instruments is the primary manufacturer for all of the TMS 9900 series products. TMS 9900 series products are handled out of the following Texas Instruments office:

TEXAS INSTRUMENTS, INC.
P.O. Box 1443
Houston, Texas 77001

Second sources for the TMS 9900 family are:

AMERICAN MICROSYSTEMS, INC.
3800 Homestead Road
Santa Clara, California 95051

SMC MICROSYSTEMS CORP. (TMS 9980 series only)
35 Marcus Blvd.
Hauppage, N.Y. 11787

THE TMS 9900 MICROPROCESSOR

The TMS 9900 is manufactured using N-channel silicon gate MOS technology. It is packaged as a 64-pin DIP. Three power supplies are required: -5V. +5V. and +12V.

Using a 3 MHz clock. instruction execution times range between 3 and 10 microseconds.

A TMS 9900 FUNCTIONAL OVERVIEW

Figure 18-1 illustrates that part of our general microcomputer system logic which is implemented by the TMS 9900 CPU.

The most important features of Figure 18-1 are:

- The absence of programmable registers
- The presence of significant interrupt handling logic
- The presence of serial-to-parallel data conversion logic
- The absence of I/O port interface logic

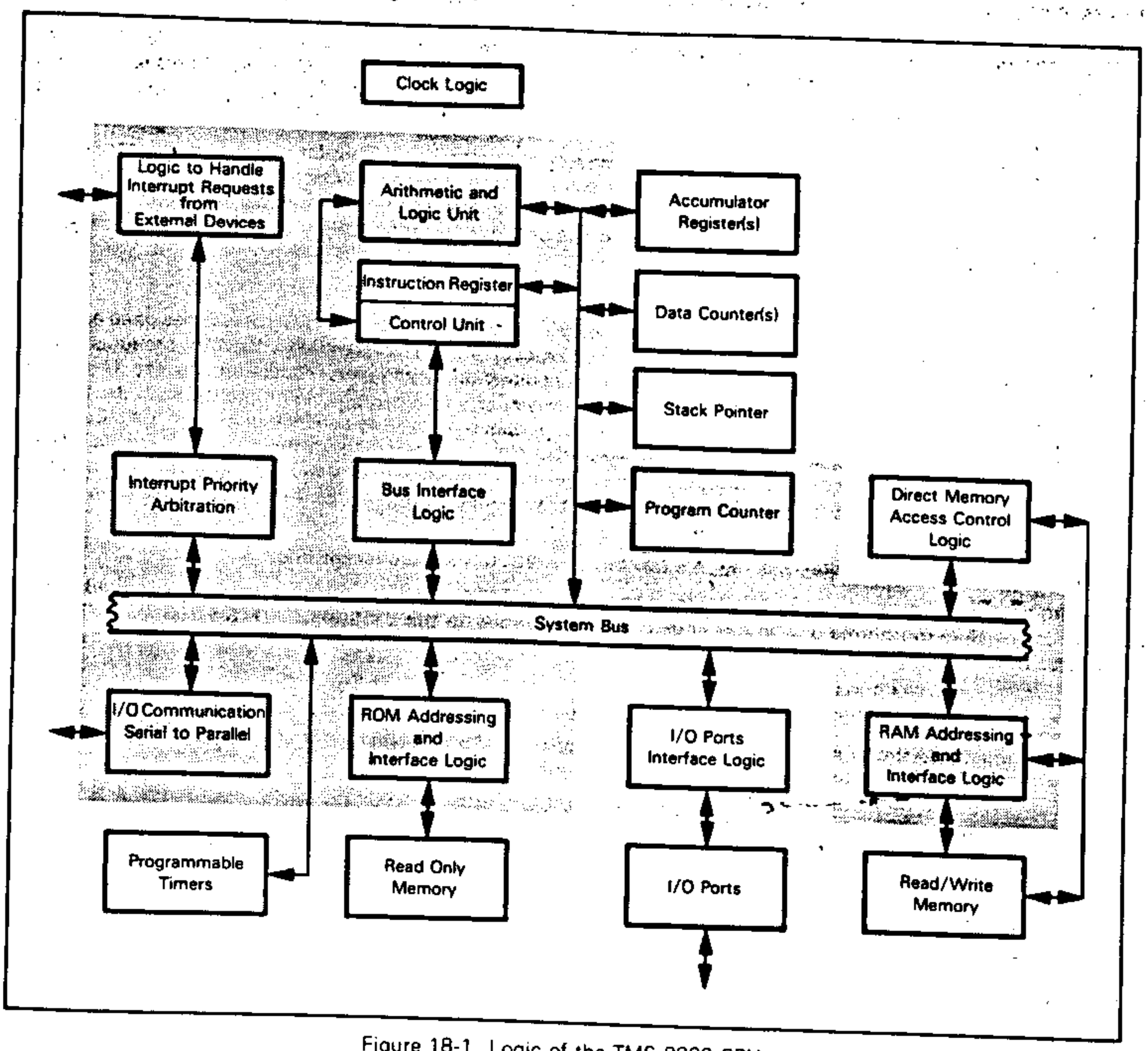


Figure 18-1. Logic of the TMS 9900 CPU

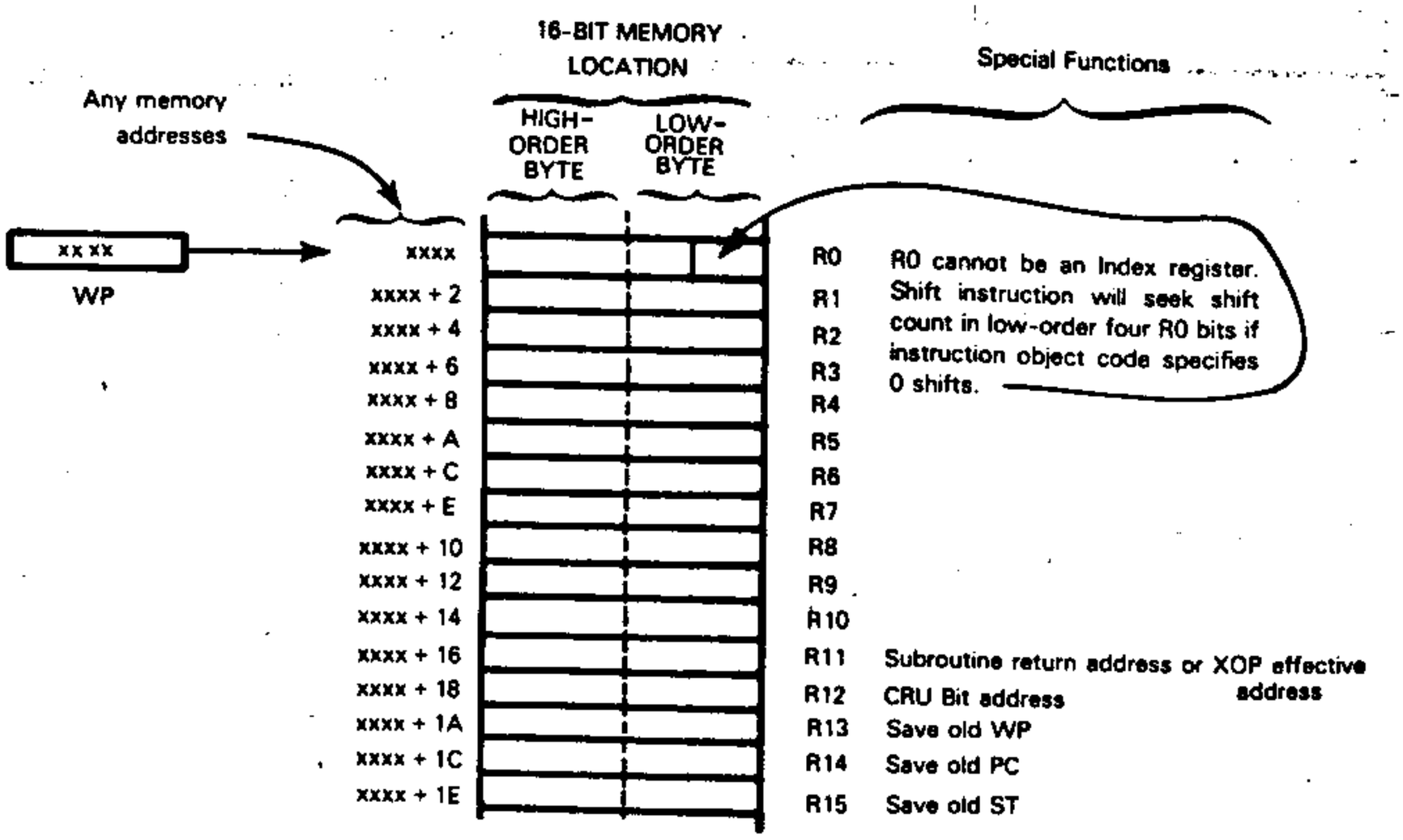
Let us first consider the manner in which the TMS 9900 handles programmable registers.

TMS 9900 PROGRAMMABLE REGISTERS

Within the logic of the TMS 9900 itself, there are just three 16-bit programmable registers: a Program Counter, a Workspace register, and a Status register.

The Program Counter and Status register are straightforward. The Program Counter contains the address of the next instruction to be executed. The Status register maintains various statuses, which we describe later in this chapter.

The Workspace register is a unique and powerful programming feature of the TMS 9900. This register identifies the first of sixteen 16-bit memory locations which act as 16 General Purpose registers. This may be illustrated as follows:

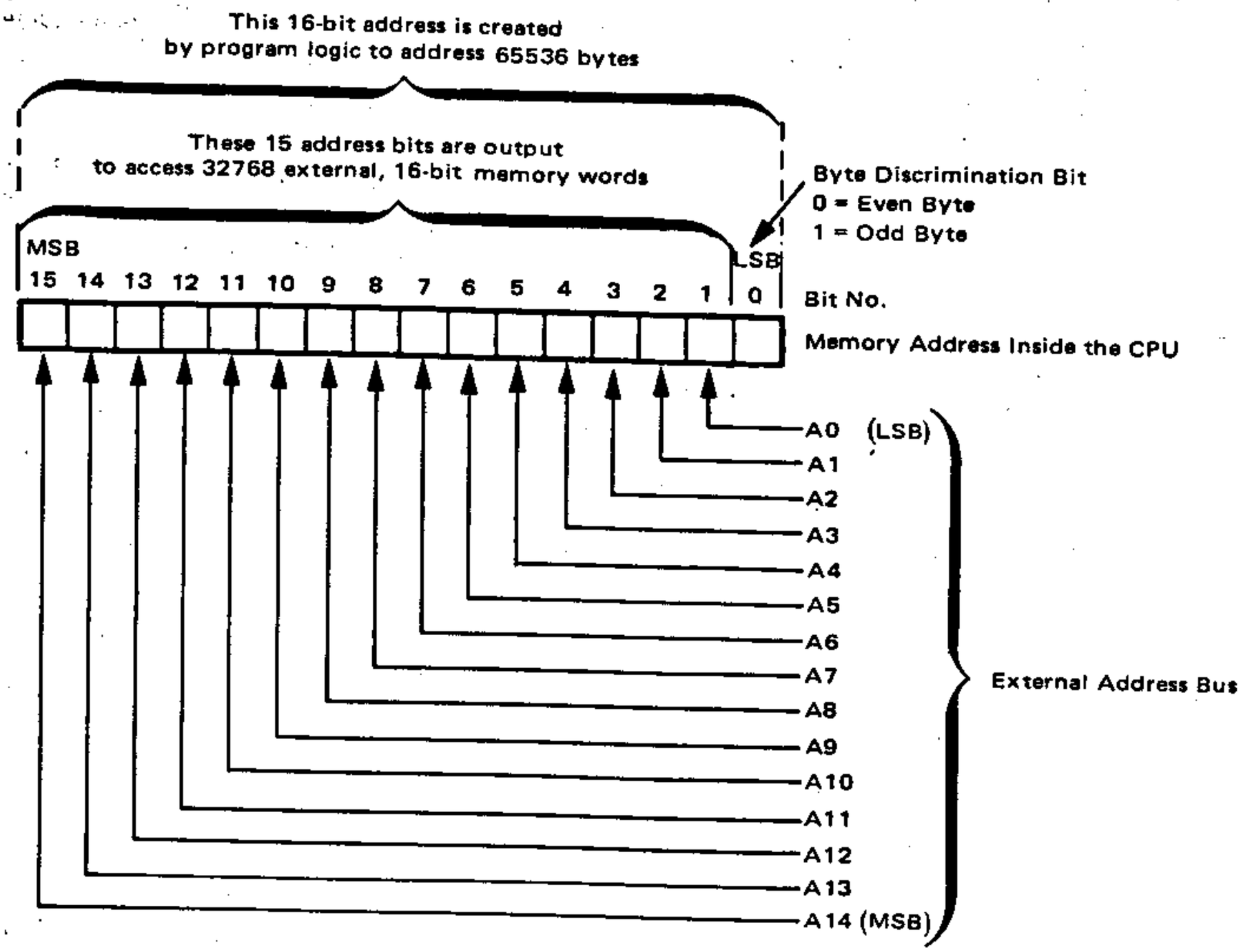


Some of the 16 registers serve special functions, as defined by the text on the right-hand side of the illustration above. For the moment, do not attempt to understand these special functions. They are described later in the chapter.

In TMS 9900 microcomputer systems, external memory consists of 16-bit memory words. Each 16-bit memory word has its own memory address. Within the TMS 9900 CPU, however, memory is addressed as a sequence of 8-bit locations. For this to occur, the CPU

TMS 9900 MEMORY ADDRESSES

generates an internal 16-bit memory address: the high-order 15 bits of the internal memory address create the external memory addresses. **This may be illustrated as follows:**



When designing hardware around the TMS 9900, you will implement external memory as 16-bit words, which are addressed by a 15-line Address Bus. That is to say, 32,768 16-bit words may be addressed.

But when you are programming the TMS 9900 you will visualize memory as 65,536 bytes, addressed by a 16-bit address. **An even byte address will access the low-order byte of an external 16-bit memory word, while an odd memory address will access the high-order byte of an external 16-bit memory word.**

Any 16 contiguous words of read/write memory may serve as the current 16 general purpose registers for the TMS 9900.

You may have as many sets of 16-bit registers as you wish, limited only by the size of implemented memory.

If you are using more than one set of 16-bit registers, then at any time just one set of 16-bit registers can be selected. The WP register identifies the first of the 16 contiguous memory locations serving as the current 16 general purpose registers.

Each of the 16 general purpose registers may be used to store data or addresses. Thus, **each general purpose register may serve as an Accumulator or as a Data Counter.**

Registers R11 through R15 are used as special Pointer storage buffers; we will be describing the way in which these registers are used as the chapter proceeds.

Having 16 general purpose registers in read/write memory, rather than in the CPU, is the single most important feature of TMS 9900 architecture. The advantage of having 16 general purpose registers located anywhere in read/write memory is that you can have many sets of 16 general purpose registers. For example, following an interrupt acknowledge, you no longer need to save the contents of general purpose registers — all you need to do is save the contents of the Program Counter, the Workspace register and the Status register, and that is done automatically by TMS 9900 interrupt handling logic. By loading new values into the Program Counter and the Workspace register, you

can begin executing a new program, accessing 16 new memory words — which will be treated as a new set of 16 general purpose registers.

The disadvantage of having 16 general purpose registers in read/write memory is that no TMS 9900 microcomputer system can be configured without read/write memory; and if you are going to use many different sets of 16-bit registers, then you are going to require a significant amount of read/write memory. Furthermore, you lose the speed associated with executing register-to-register operations: there are no source and destination locations left in the CPU. Every register access becomes a memory access.

TMS 9900 CONTEXT SWITCH

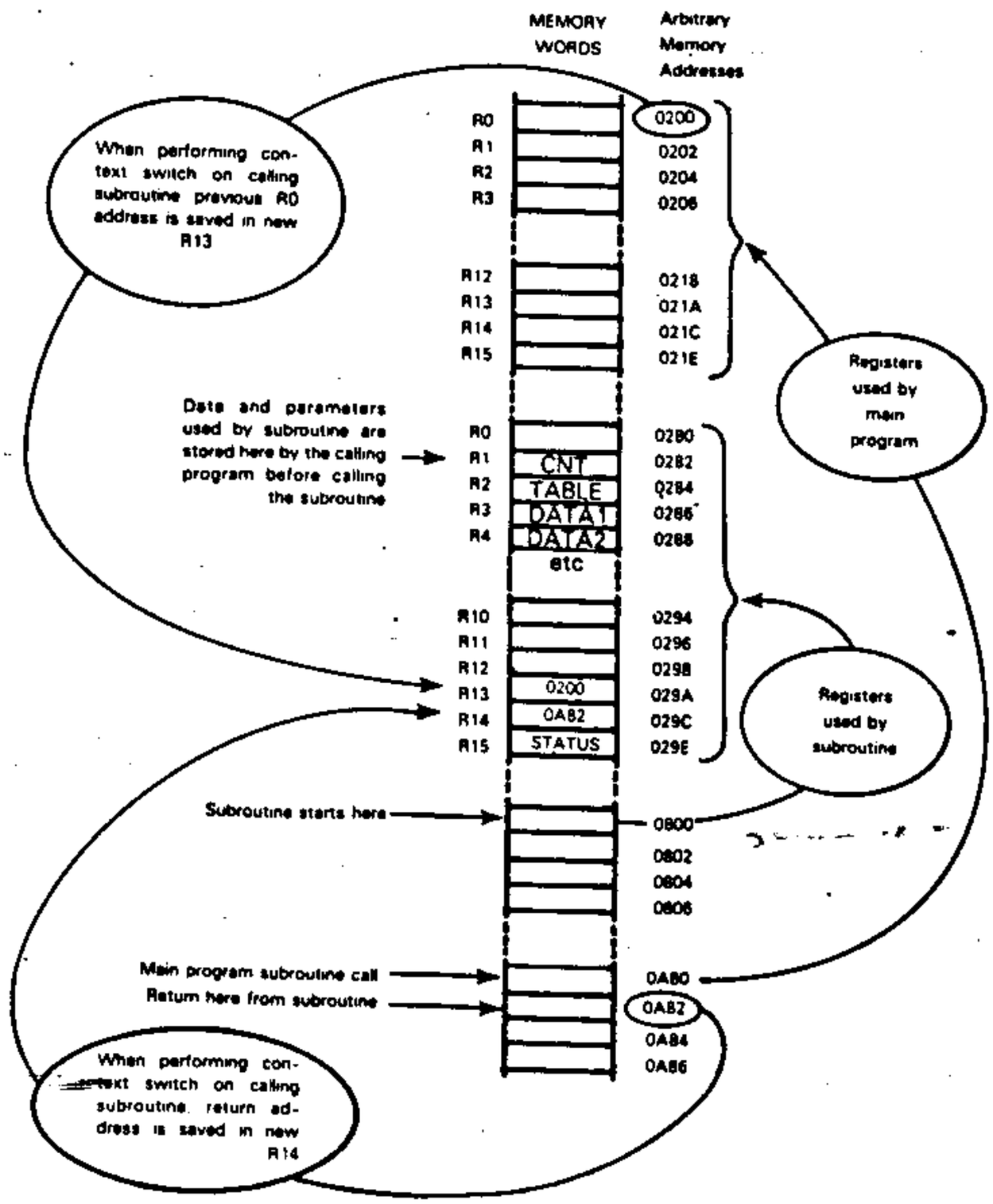
TMS 9900 literature refers to the process of switching from one set of general purpose registers to another as a context switch. This terminology reflects the complete change of program environment that results from the switch.

Special instructions allow you to perform a forward context switch or a backward context switch.

During a forward context switch, you load new values into the Workspace register and Program Counter, while simultaneously saving the old Workspace register, Program Counter, and Status register contents in the new General Purpose Registers R13, R14, and R15.

A backward, or reverse context switch loads the current contents of General Purpose Registers R13, R14, and R15 into the Workspace register, Program Counter, and Status register, respectively, thus returning you to your previous set of general purpose registers.

You can perform context switches as often as you like and whenever you like. For example, a very effective way of using context switching is to group data into contiguous memory words which you can identify as a register set. Upon entering a subroutine, you can perform a context switch which automatically creates all necessary initial data and address values in appropriate general purpose registers. This may be illustrated as follows:



As illustrated above, **when you perform a forward context switch,** the current Program Counter contents, Status register contents, and WP register contents are saved in what will become the new Registers R13, R14 and R15, respectively. **Here is the exact sequence in which events occur:**

TMS 9900 FORWARD CONTEXT SWITCH

1) The new WP register contents are loaded into the CPU and held in temporary storage.
2) The current Status register contents are written out to the memory location which will become the new Register R15.
3) The current Program Counter contents are written out to the memory location which will become the new Register R14.
4) The current WP register contents are written out to the memory location which will become the new Register R13.
5) The new WP register contents, which were held in temporary storage, are moved into the WP register.
6) The new value is loaded into the Program Counter.

Thus, when a forward context switch is performed, an audit trail ensures that program logic knows the exact machine state at the instant of the forward context switch.

TMS 9900 BACKWARD CONTEXT SWITCH

When a backward context switch occurs, the contents of the current General Purpose registers R13, R14, and R15 are loaded into the WP register, the Program Counter, and the Status register, respectively. Thus, program logic returns to the location of the forward context switch.

TMS 9900 MEMORY ADDRESSING MODES

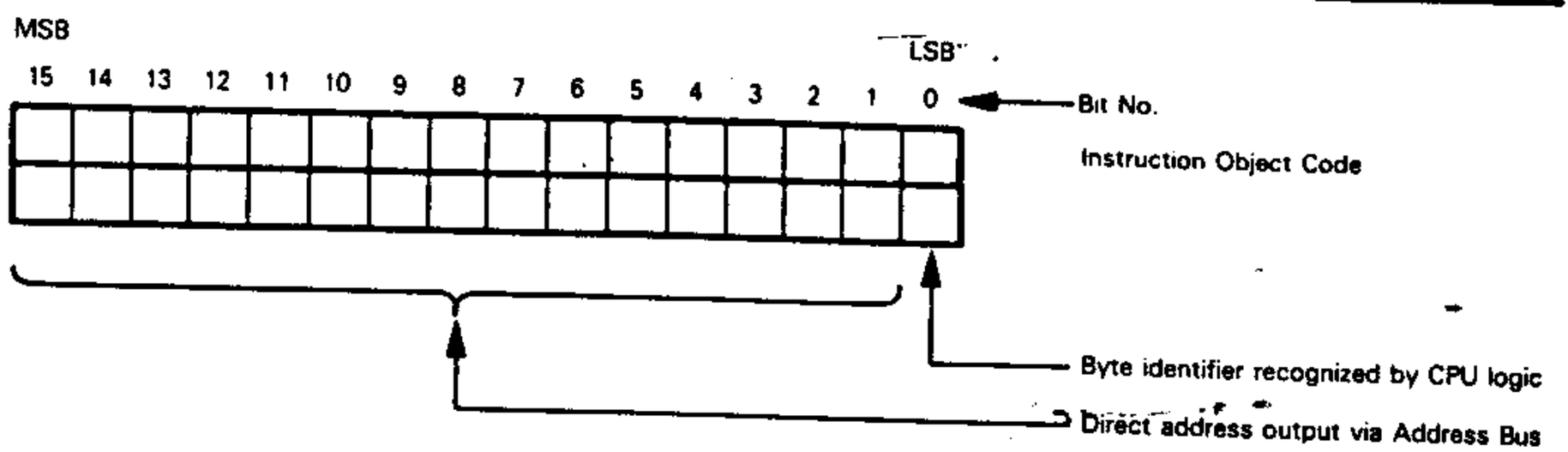
The TMS 9900 provides these four methods of addressing memory:

1) **Direct memory addressing**
2) **Direct, indexed memory addressing**
3) **Implied memory addressing**
4) **Implied memory addressing with auto-increment**

The way in which the TMS 9900 implements these four memory addressing modes is exactly as described in Volume 1, Chapter 6. The important point to note is that the TMS 9900 looks upon its address space as consisting of 32,768 16-bit memory words which are addressed using 15, rather than 16, Address Bus lines, yet programs compute all addresses as 16-bit words. This logic was described earlier.

TMS 9900 DIRECT ADDRESSING

Direct memory addressing instructions provide the memory address in the second word of an instruction's object code:



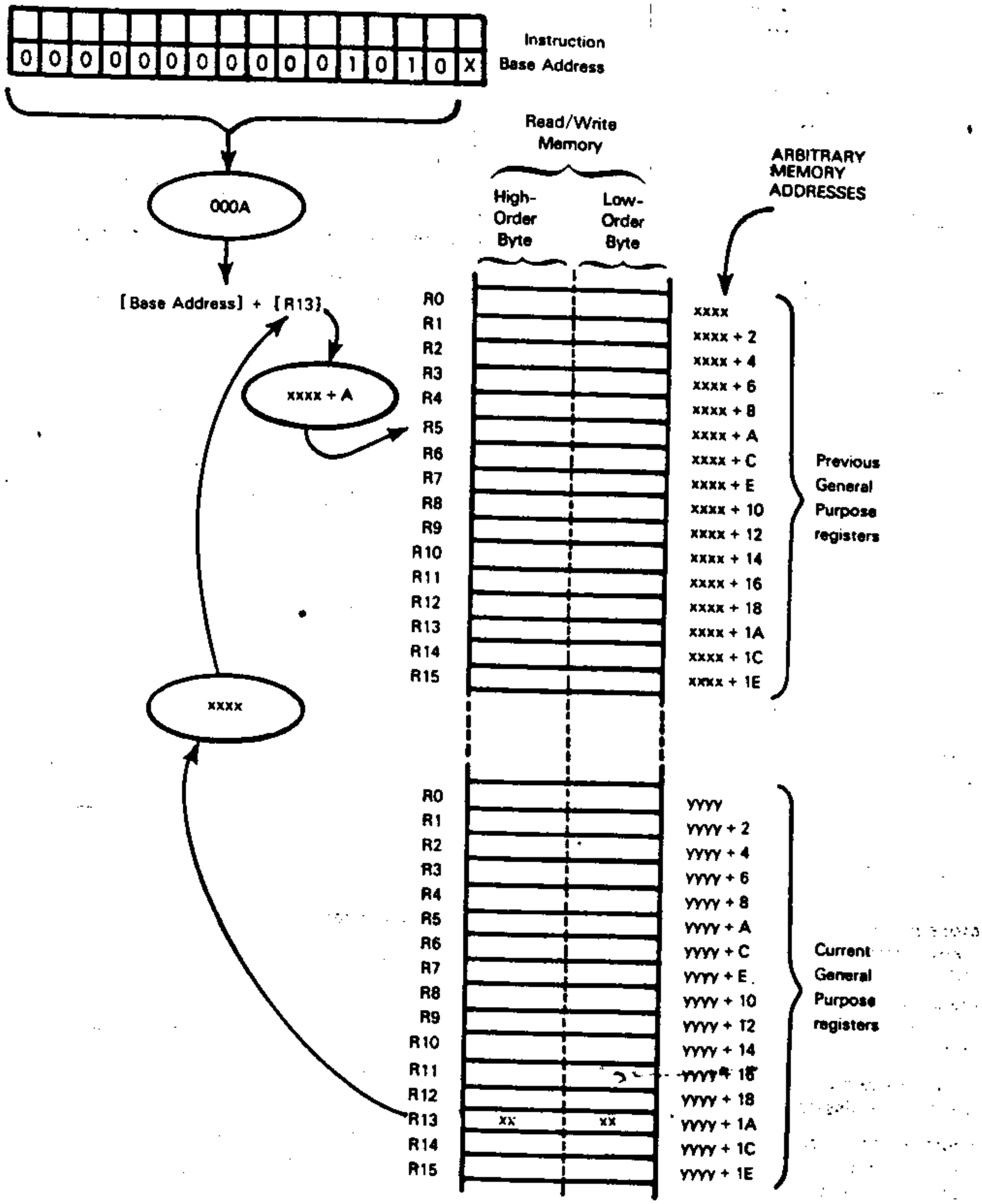
TMS 9900 INDEXED ADDRESSING

Direct, indexed memory addressing instructions provide a base address in the second object code word, but they also identify a general purpose register whose contents are to be added, as a signed binary number, to the base address. Again, the low-order bit of the computed address is not output via the Address Bus, but is interpreted by CPU logic as a byte identifier.

General Purpose Register R0 cannot be specified as an index register.

Direct, indexed addressing is very useful in a TMS 9900 microcomputer system. It allows you to address the previous set of general purpose registers, following a context switch, without knowing where the previous registers were. Suppose you want to access the contents of the memory word which was being used as General Purpose Register R5

before you switched to your current set of general purpose registers. Recall that the previous Workspace register contents are stored in your current General Purpose Register R13. You could thus address the previous General Purpose Register R5. without knowing where this general purpose register may have been. by using direct. indexed addressing as follows:

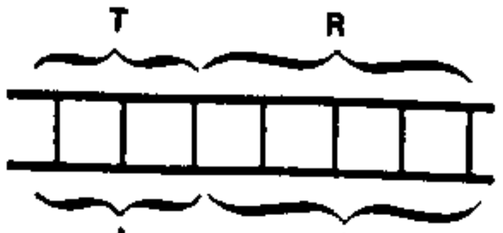


An implied memory addressing instruction will specify one of the 16 current general purpose registers as providing the effective memory address.

TMS 9900 IMPLIED ADDRESSING

If you specify implied memory addressing with auto-increment, then the contents of the identified general purpose register will be incremented after the memory access has been performed. If the instruction specifies a byte operation. the register contents will be incremented by one; the register contents will be incremented by two after a full-word operation.

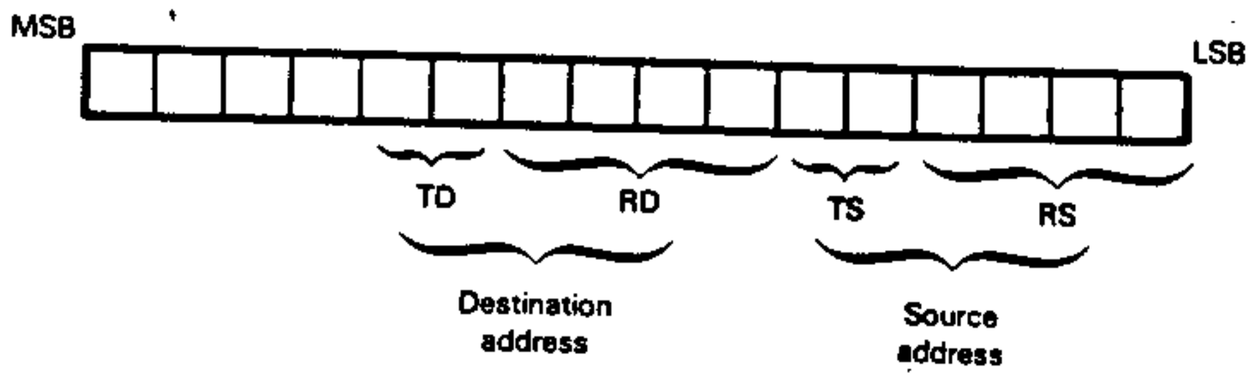
Six object code bits identify the data memory addressing option selected by any TMS 9900 instruction that accesses data memory. The six object code bits are interpreted as follows:



0000 through 1111 select the general purpose register to be accessed during the memory address computation

00 - Not a memory reference instruction. The selected register is accessed directly.
01 - Implied memory addressing
10 - Direct addressing if register R0 is selected.
Direct, indexed addressing otherwise.
11 - Implied memory addressing with auto-increment

Two-address instructions will include 12 memory addressing option bits:



Two-address instruction object code

Some instructions allow a source to be anywhere in memory, but the destination must be a general purpose register. These object codes include TS, RS, and RD, but not TD.

TMS 9900 Jump instructions use program relative, direct addressing. These are one-word instructions, where the low-order byte of the instruction object code provides an 8-bit, signed binary value, which is added to the incremented contents of the Program Counter. This is straightforward program relative, direct addressing.

TMS 9900 PROGRAM MEMORY ADDRESSING

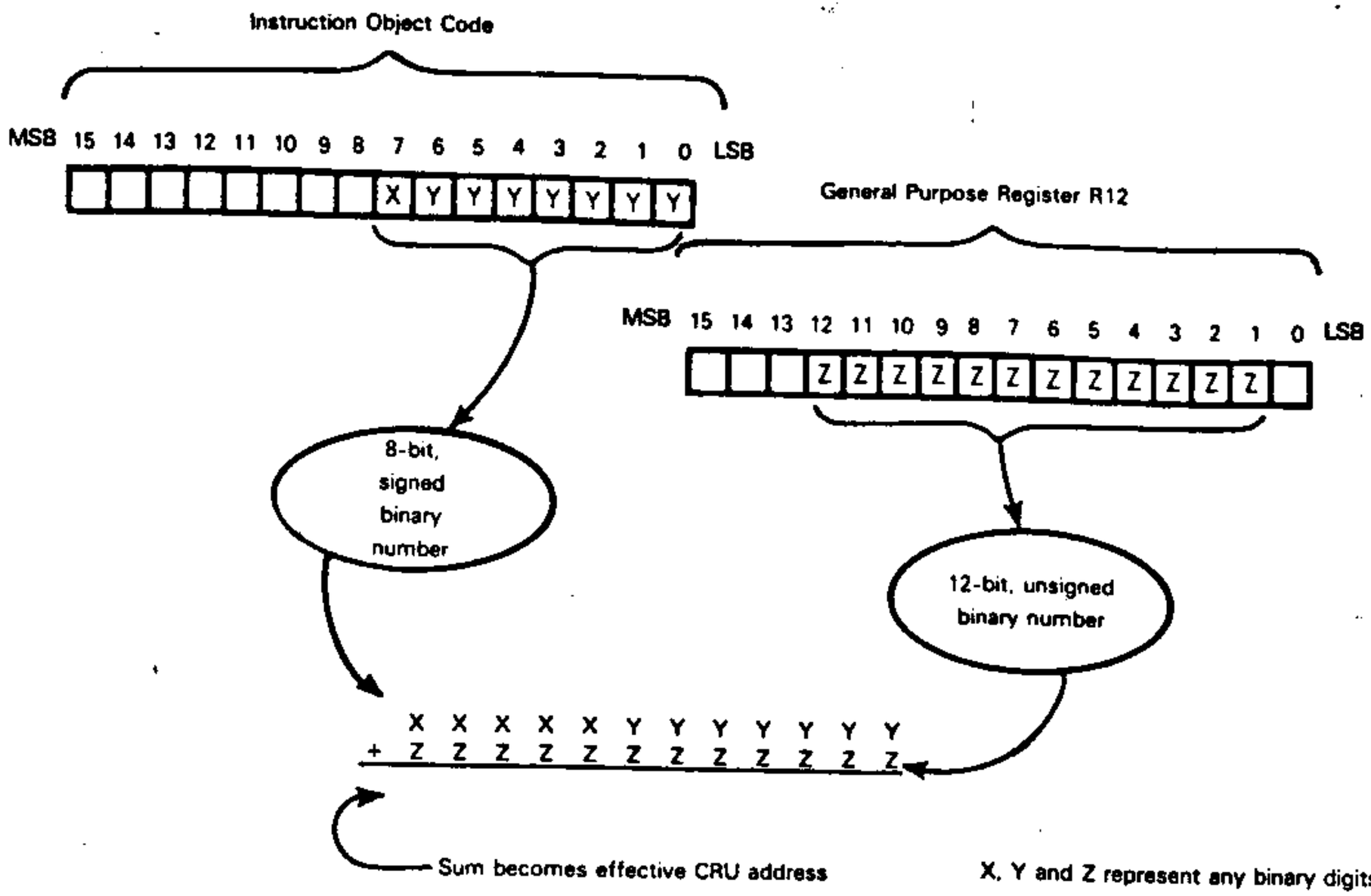
TMS 9900 I/O ADDRESSING

As compared to other microcomputers described in this book, the TMS 9900 has unusual I/O logic. **In addition to addressing I/O devices as memory locations, you can address a separate I/O field of up to 4096 bits. Texas Instruments' literature refers to this field as the "Communications Register Unit" (CRU).** If you are programming a TMS 9900 microcomputer system that has already been configured by Texas Instruments, then it is justifiable to look upon the Communications Register Unit as a form of I/O port. If you are building your own interface to a TMS 9900 CPU, then instructions that are supposed to access the Communications Register Unit in reality simply make alternative use of part of the Address Bus in conjunction with three control signals: CRUCLK, CRUIN, and CRUOUT.

There are two classes of TMS 9900 CRU instructions. The first class accesses individual bits (or signals), while the second class accesses bit fields that may be between 1 and 16 bits wide.

There are three single-bit CRU instructions; they set, reset, or test the identified CRU bit. This is equivalent to setting, resetting, or testing an external signal or single I/O port bit. When a bit is to be set or reset, the new level is output via CRUOUT, and a CRUCLK pulse indicates that valid data is on the CRUOUT line. When the condition of a bit is to be input or tested, then external logic is required to return the level of the tested bit via CRUIN.

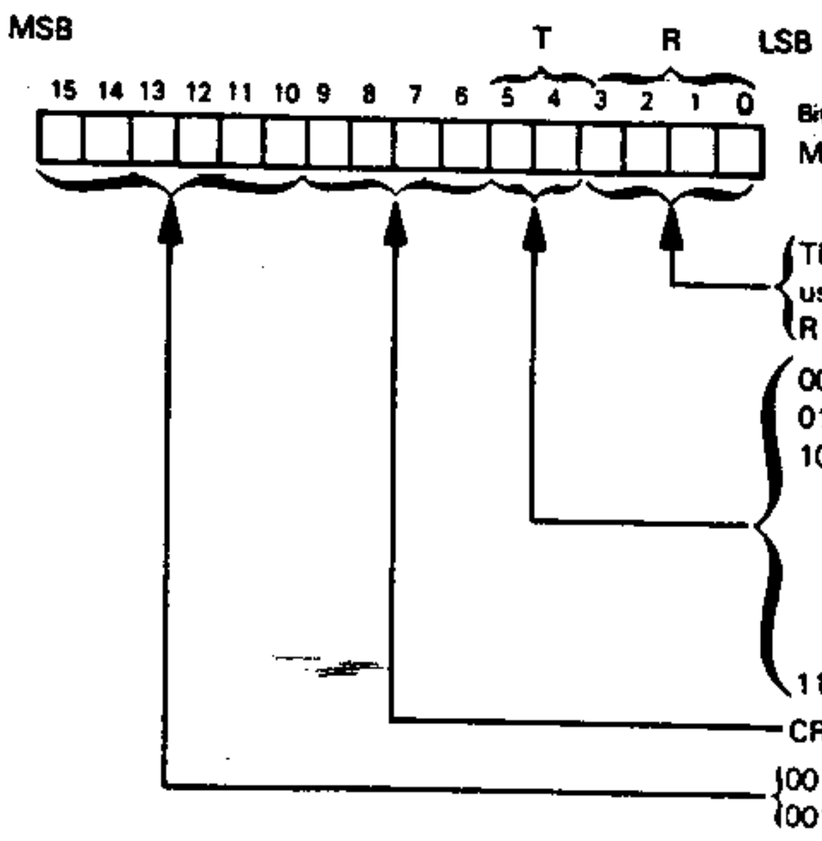
A CRU bit instruction outputs a 12-bit address which is computed as follows:



The 12-bit address is output on the 12 lower-order address lines; the three higher-order address lines are all 0 to designate a CRU address.

Now during the execution of a CRU bit instruction. the address which is output is supposed to be a bit address — that is. an address identifying one bit in a possible 4096-bit field. So far as external interface logic is concerned. the address can be interpreted in any way. However. **data output will occur via CRUOUT only; data is input via CRUIN, and stored in the Equal bit of the Status register.**

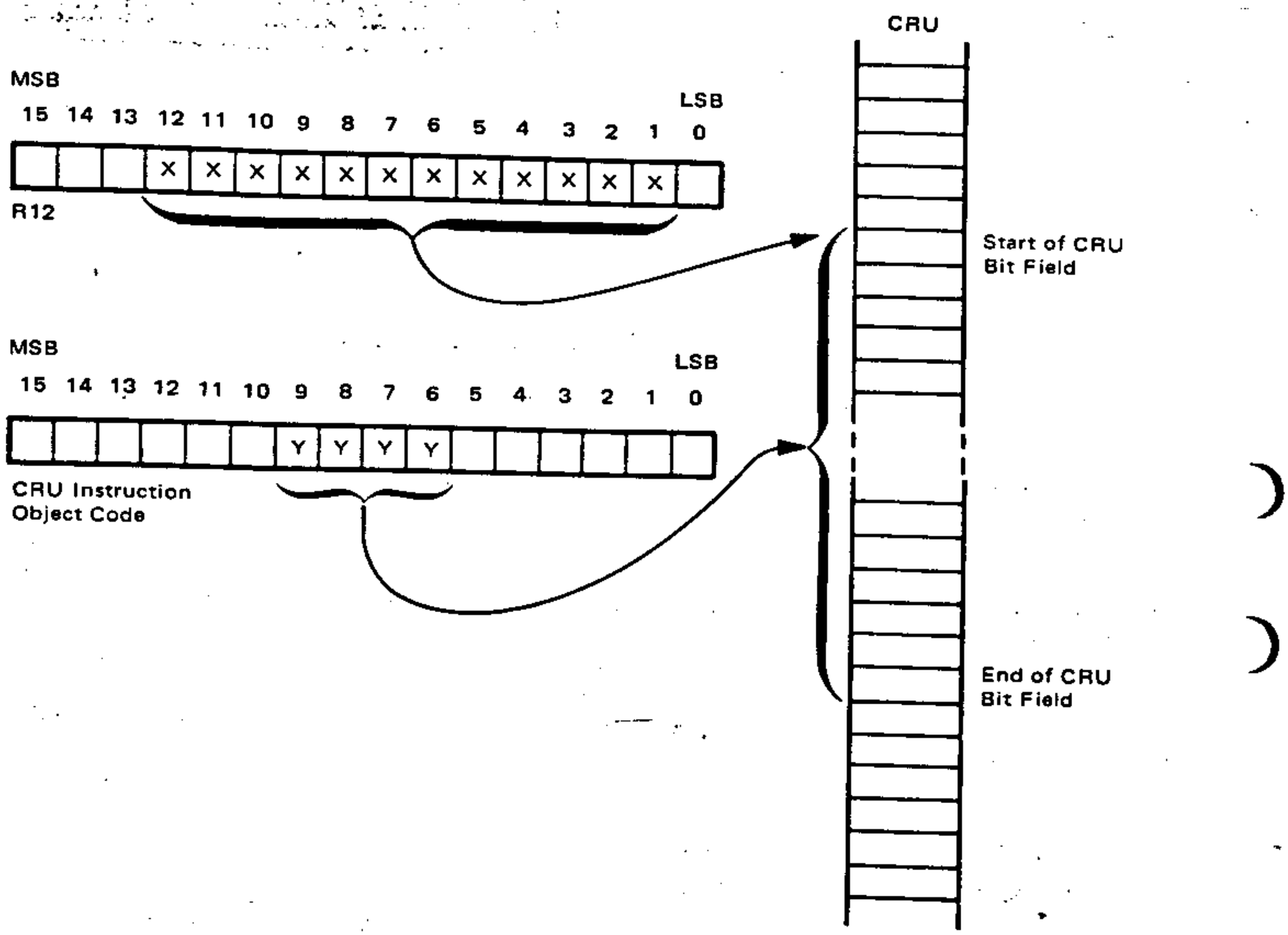
There are two multi-bit CRU instructions: one. LDCR. transfers data from an addressed memory location to any addressed CRU bit field. The other. STCR. transfers data from an addressed CRU bit field to any addressed memory location. Anywhere from 1 to 16 bits of data may be transferred by the LDCR-and STCR instructions. **Instruction object codes are interpreted as follows:**



The source/destination memory location is identified as it would be for any memory reference instruction.

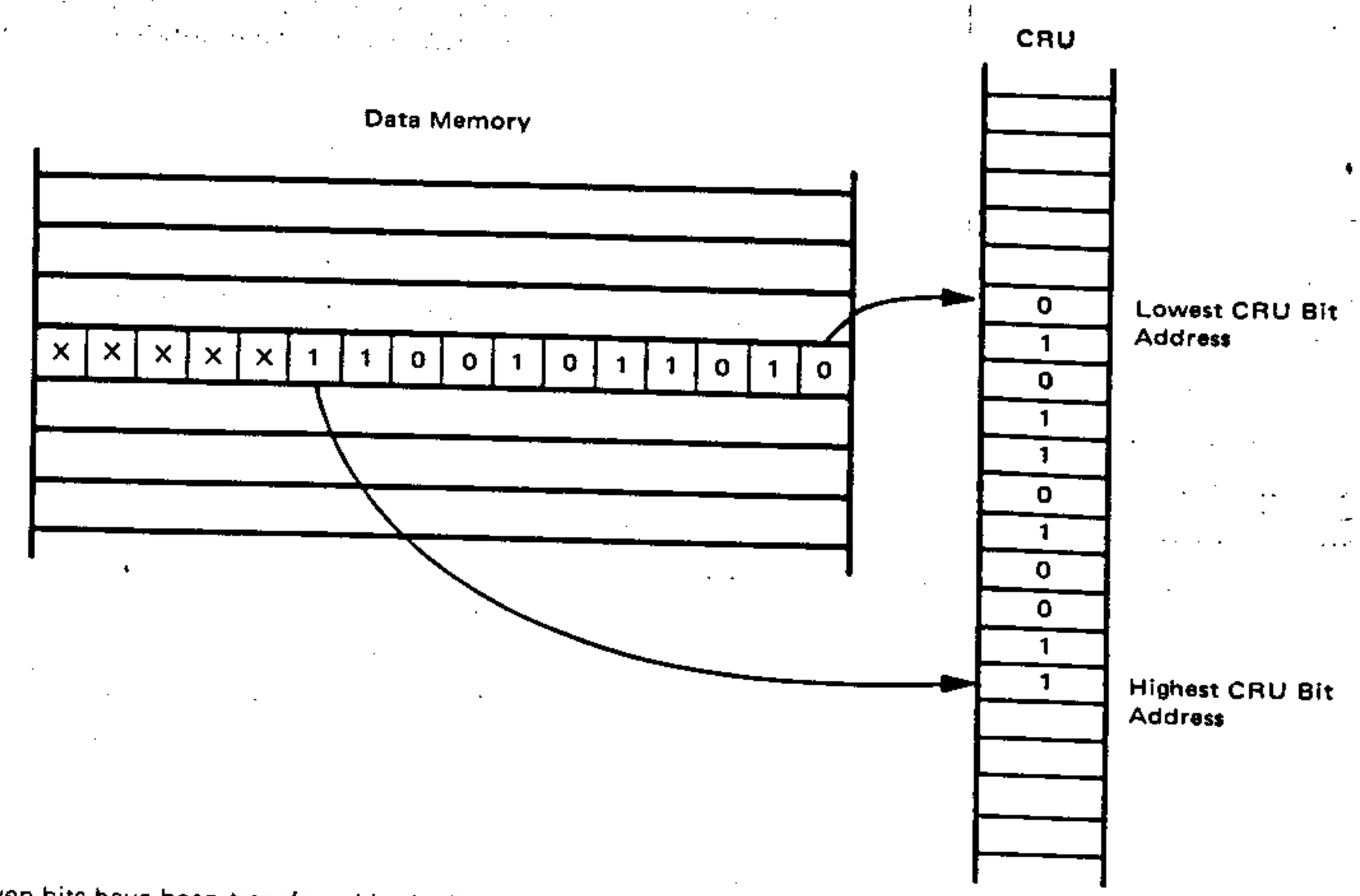
The address of the first CRU bit is specified by Register R12. For a multi-bit CRU instruction, the CRU bit address is incremented for each succeeding bit access, but the incremented address is held in a temporary storage location. The contents of Register R12 are not incremented.

Thus, multi-bit CRU instructions may transfer anywhere from 1 to 16 bits between any memory location and any CRU bit field. **Note that memory must be divided into 16-bit words, each of which has identified bit boundaries, but there are no equivalent bit boundaries in the CRU bit field.** That is to say, any CRU bit may be identified via Register R12 as the first bit in a multi-bit field, while the length of the multi-bit field is identified by the instruction object code. This may be illustrated as follows:

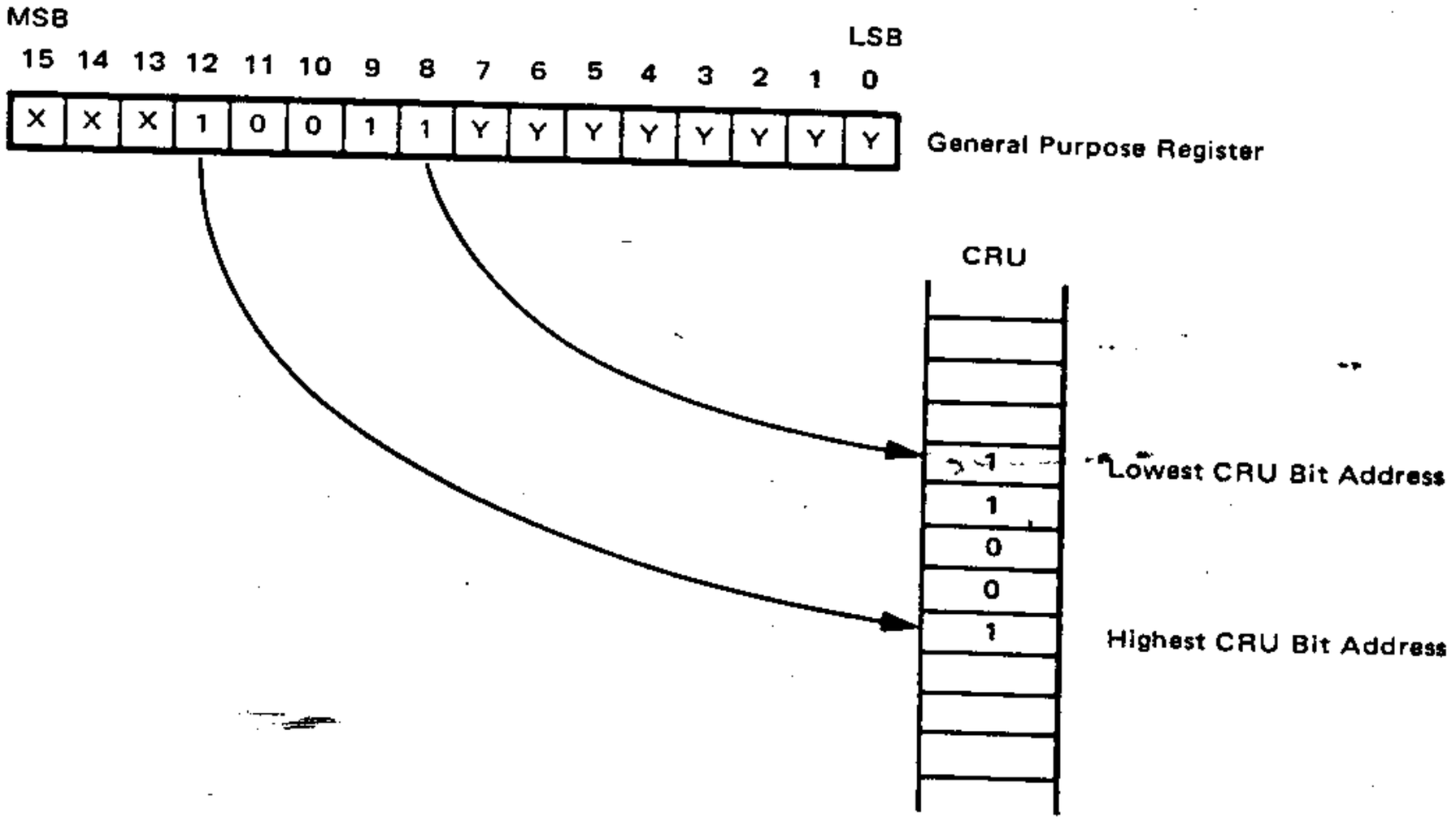


If YYYY is 0000, the CRU bit field is assumed to be 16 bits in length.

When bits are transferred from a memory location to a CRU bit field, the contents of the memory location are not actually modified, but **the transfer occurs as though bits had been right shifted out of the memory location.** Bits arriving within the addressed CRU bit field are stored in sequential CRU bit locations with ascending addresses. This may be illustrated as follows:

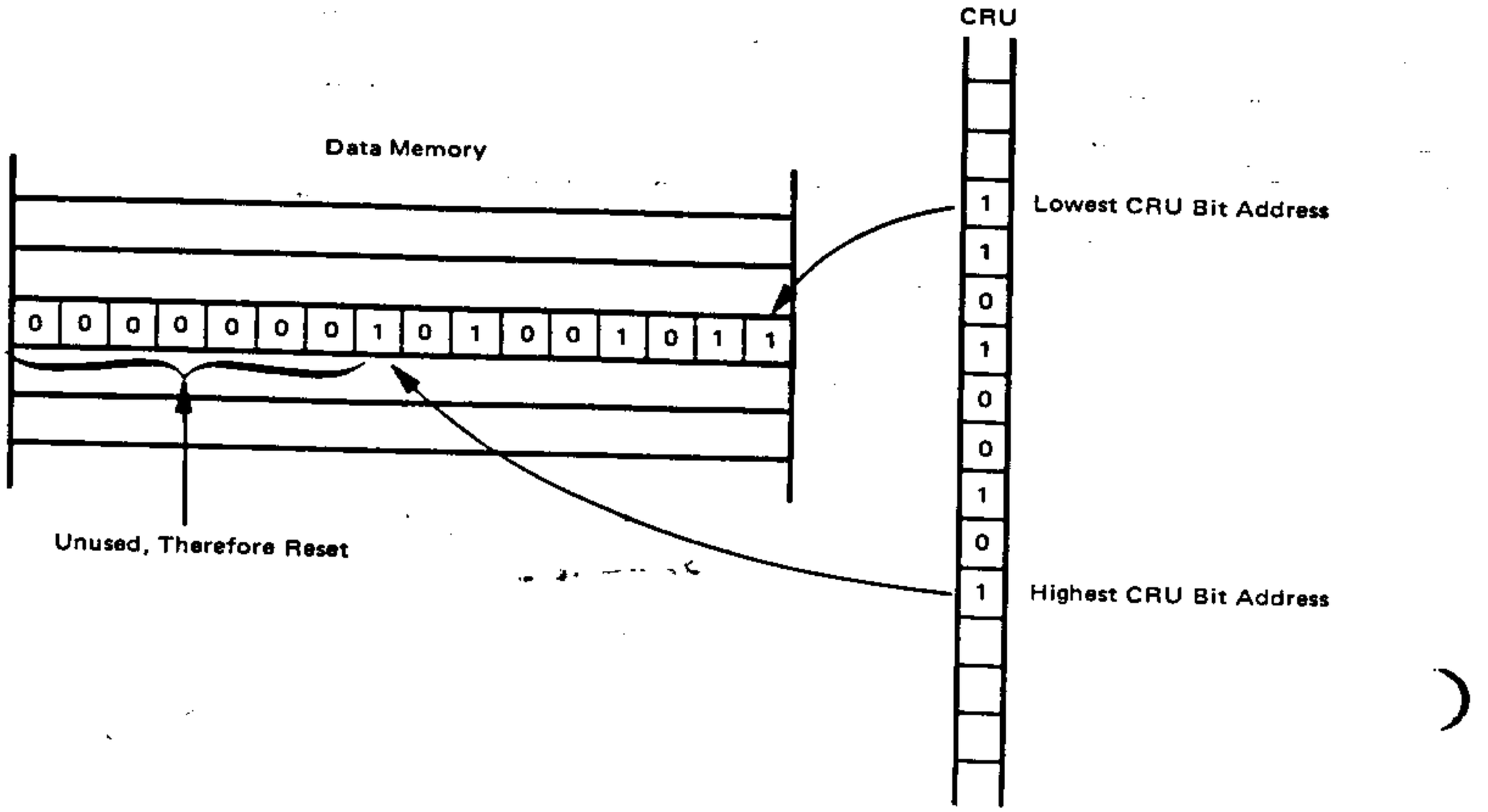


Eleven bits have been transferred in the illustration above. If eight or fewer bits are transferred from a general purpose register. only the more significant byte is accessed:

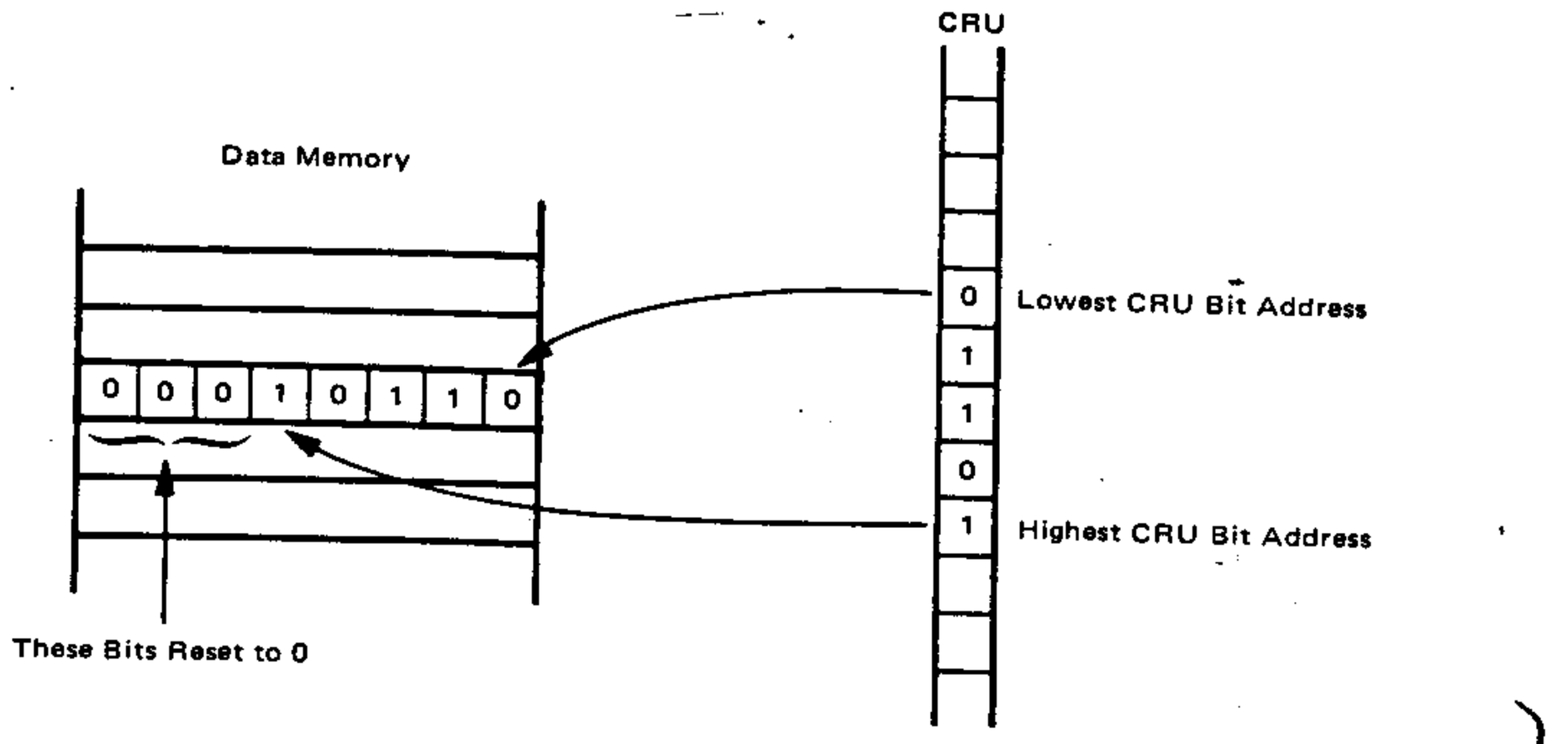


Our illustration shows a transfer of five bits.

...from data memory word are zeroed if unfilled. This **may be illustrated as follows:**

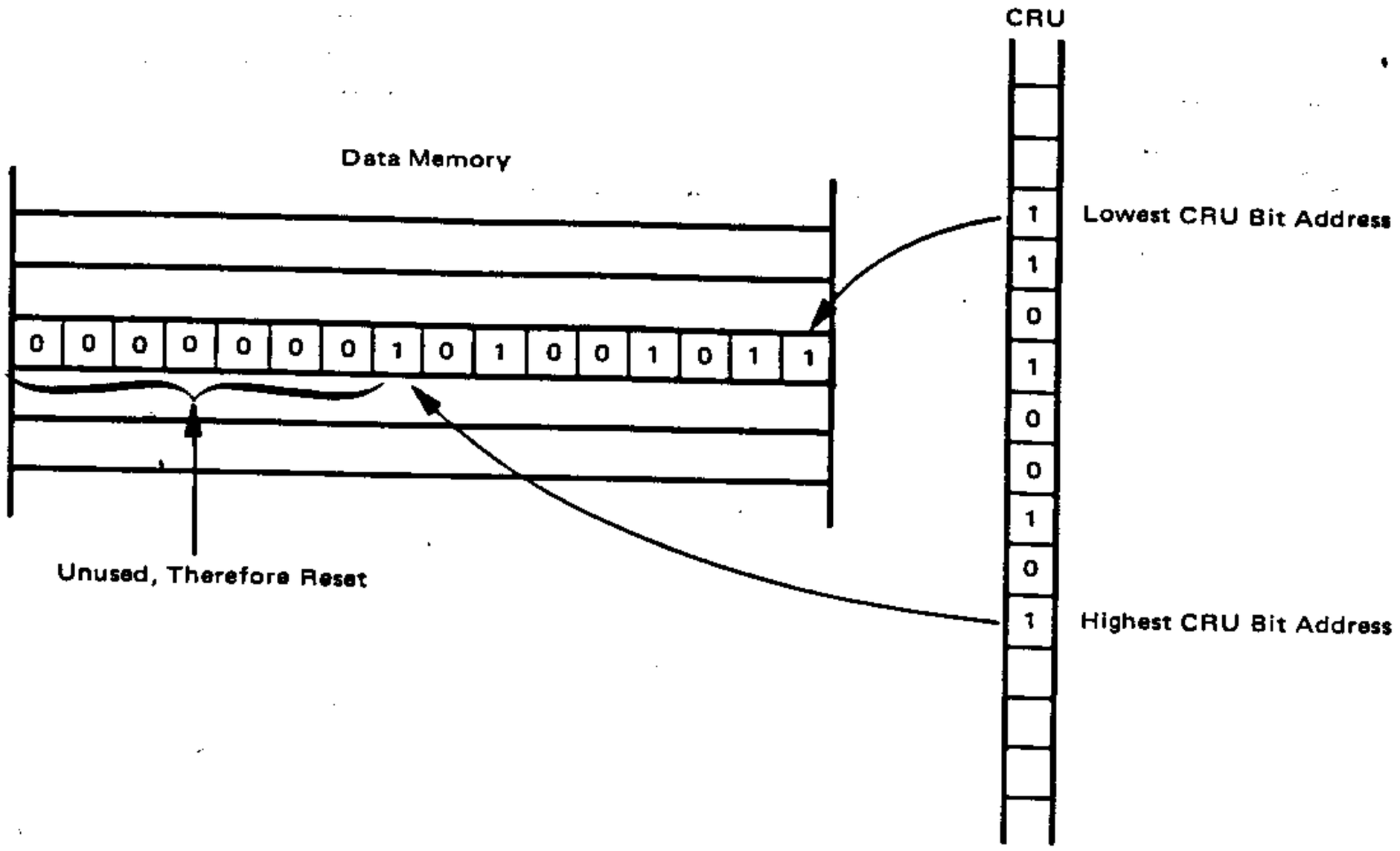


As with data transfers from memory to the CRU. if eight or fewer bits are transferred. only a byte will be affected. This will be either the addressed memory byte:

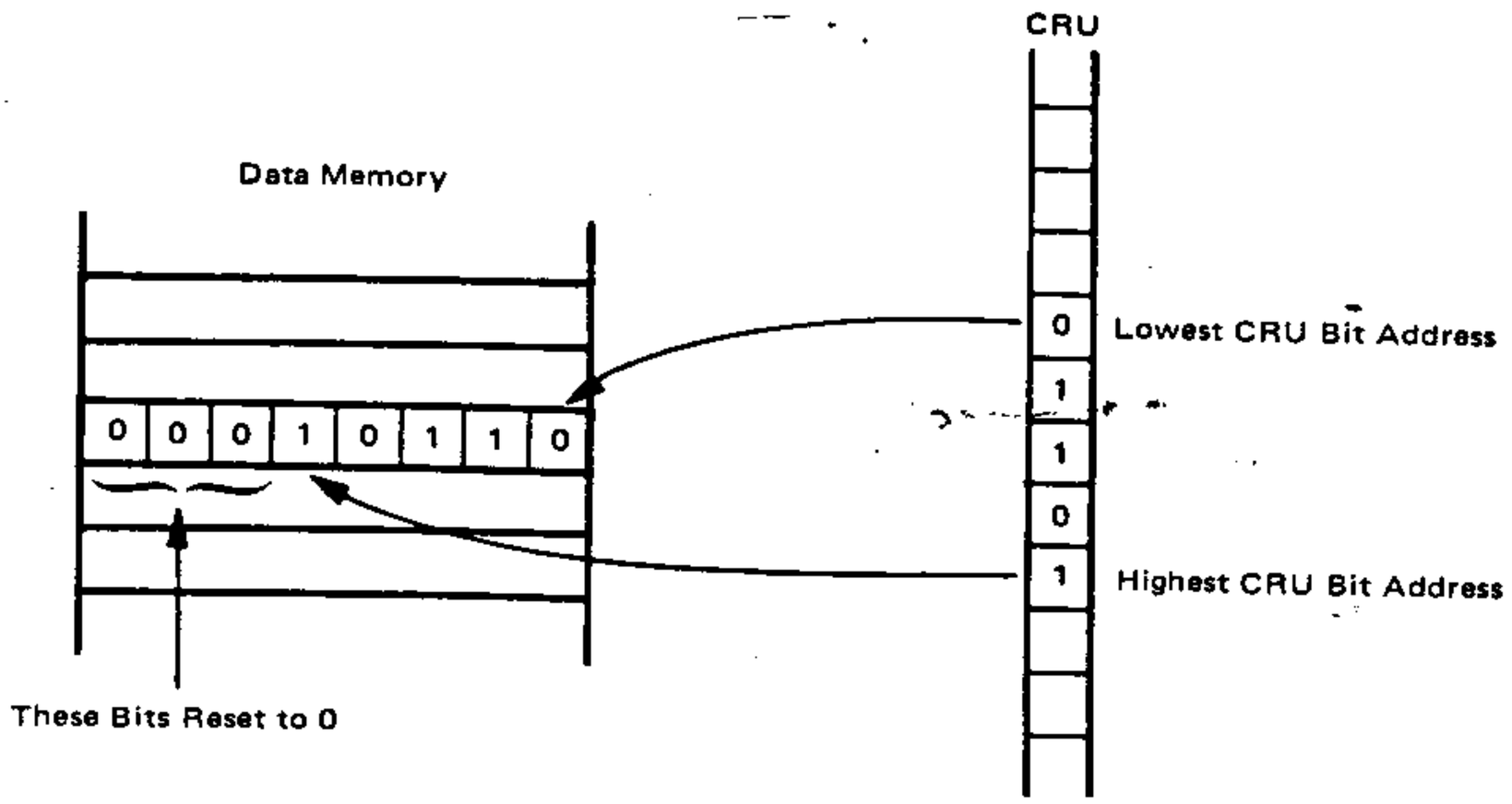


If eight or fewer bits are transferred from a memory location, then the memory address will be considered a byte address rather than a word address; that is, the transfer will be from the low-order bits of the addressed byte, which may be either the upper or lower byte of a 16-bit memory word. Thus you can access the lower byte of a general purpose register by addressing it as a memory location.

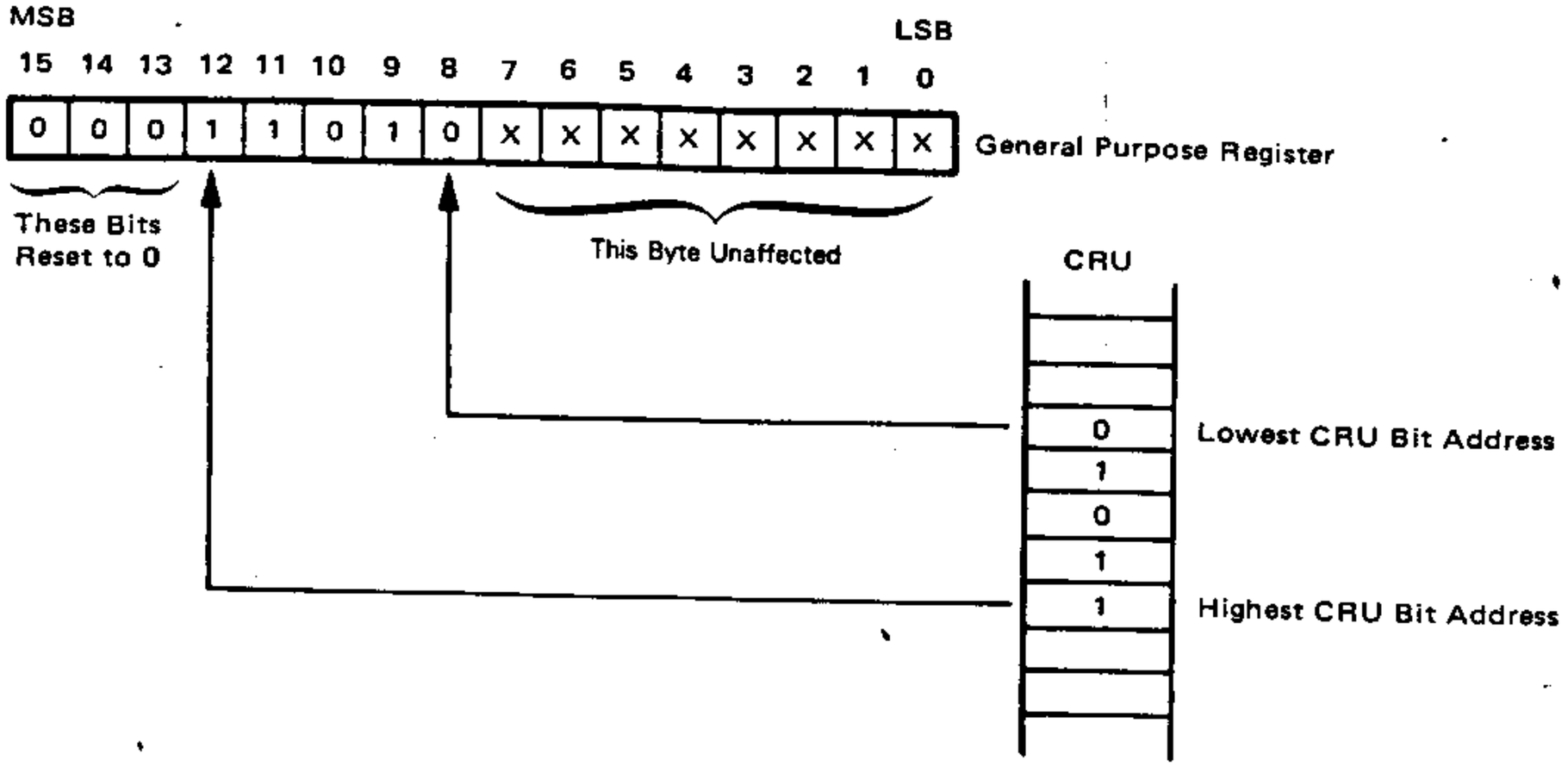
A data transfer from the CRU to data memory occurs as the exact logical reverse of the illustration above, except that high-order bits of the destination data memory word are zeroed if unfilled. This **may be illustrated as follows:**



As with data transfers from memory to the CRU, if eight or fewer bits are transferred, only a byte will be affected. This will be either the addressed memory byte:

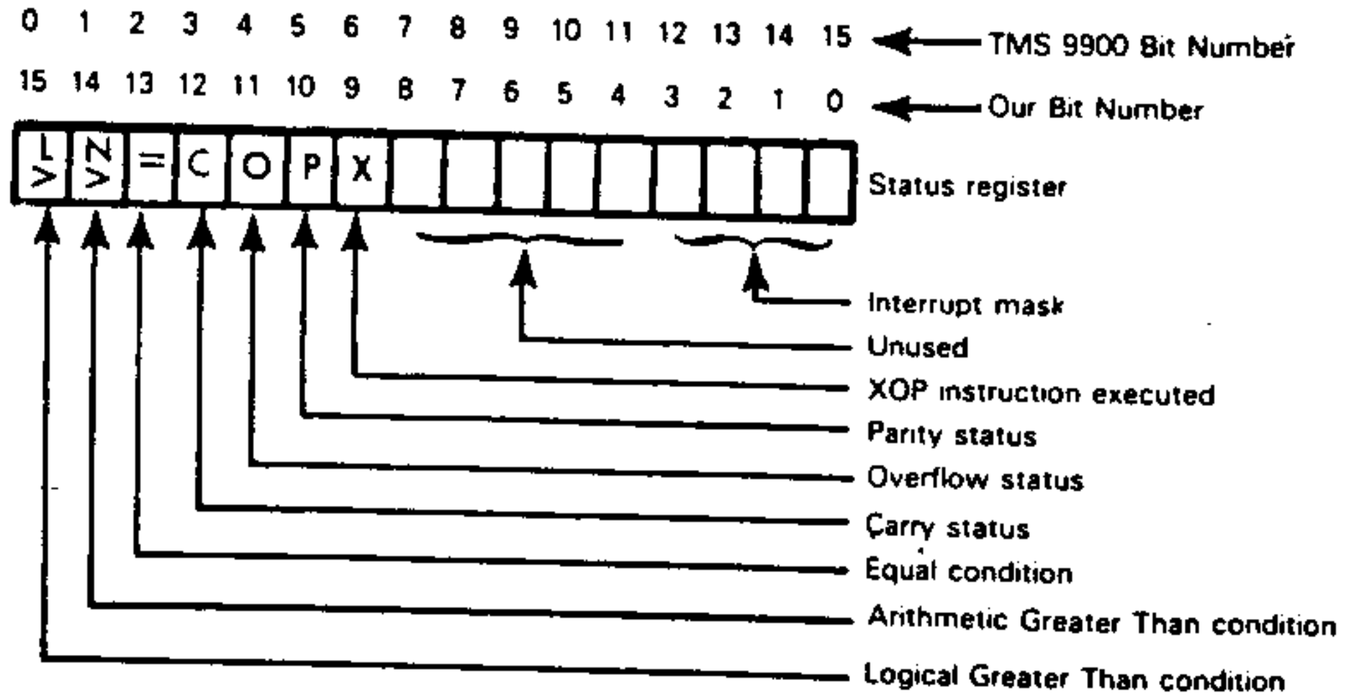


or the high-order byte of a general purpose register:



TMS 9900 STATUS FLAGS

The TMS 9900 CPU has a 16-bit Status register which may be illustrated as follows:



The low-order four bits of the Status register represent an interrrupt mask which identifies the level of interrupt which is currently enabled. As the 4-bit interrupt mask would imply, 16 levels of interrupt are allowed. We will describe interrupt processing later in this chapter.

The X status is set to 1 while an XOP instruction is being executed. This instruction allows you to perform a software interrupt — as described later in this chapter.

The P, O, and C are standard Parity, Overflow and Carry statuses.

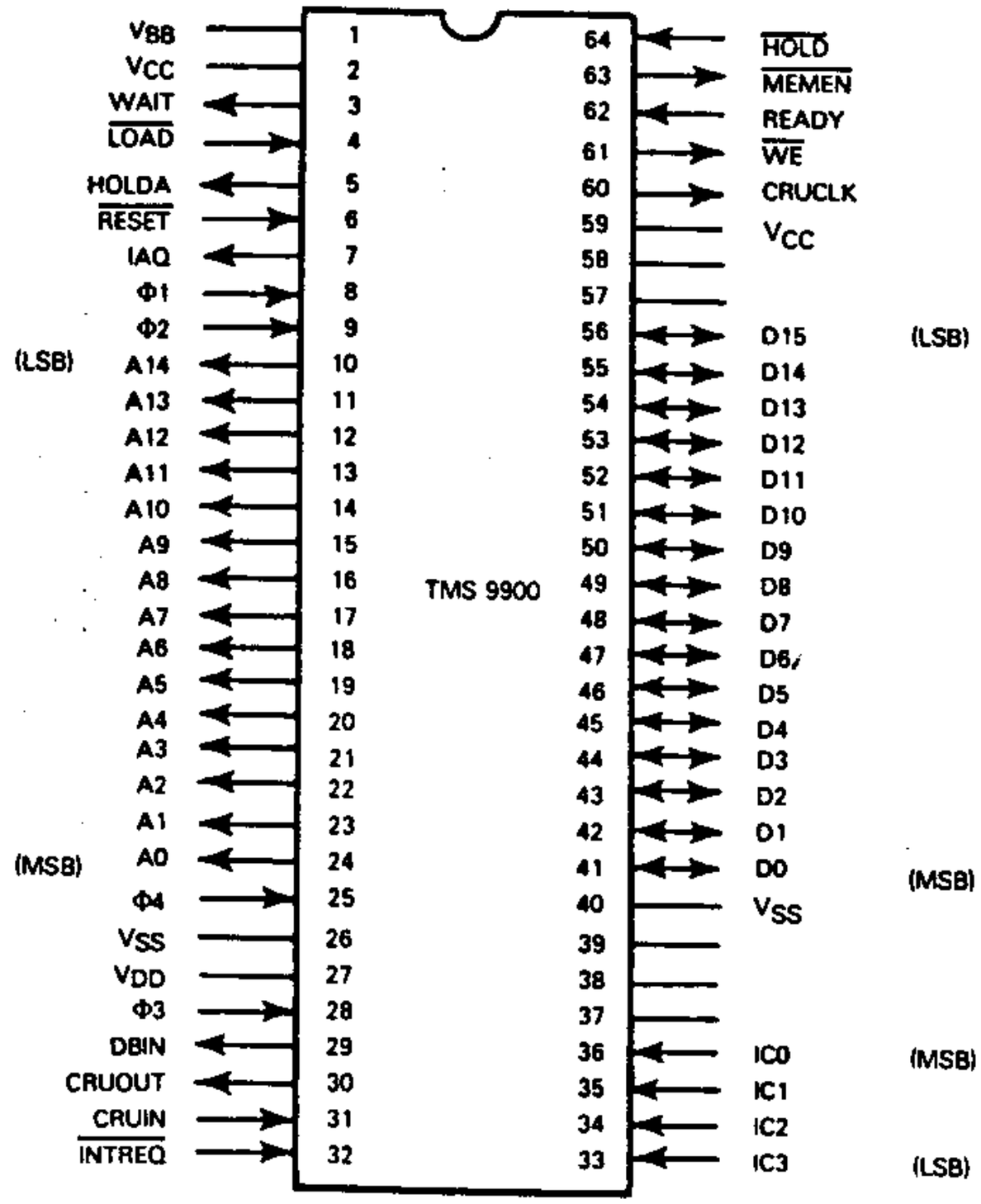
The Equal status (=) identifies a condition that currently exists, as the result of the execution of a previous instruction, that will cause a Branch-if-Equal instruction to branch. A CRU bit to be tested also gets stored in the Equal status.

The Logical Greater Than and Arithmetic Greater Than statuses are set or reset following arithmetic, logical, or data move operations. **A Logical Greater Than treats the source data as simple, unsigned binary numbers. An Arithmetic Greater Than interprets the operand as signed binary numbers.**

TMS 9900 CPU PINS AND SIGNALS

Figure 18-2 illustrates the pins and signals of the TMS 9900 CPU.

Being a 64-pin DIP, the TMS 9900 can afford to have separate Address and Data Busses.



Pin Name	Description	Type
A0 - A14	Address Bus	Tristate, output
D0 - D15	Data Bus	Tristate, bidirectional
Φ1, Φ2, Φ3, Φ4	Clock Signals	Input
MEMEN	Memory Enable	Tristate, output
IAQ	Instruction Fetch	Output
DBIN	Data Bus In	Tristate, output
WE	Write Enable	Tristate, output
READY	Memory Ready	Input
WAIT	Wait State Indicator	Output
CRUCLK	I/O Clock	Output
CRUOUT	Serial I/O Out	Output
CRUIN	Serial I/O In	Input
INTREQ	Interrupt Request	Input
IC0 - IC3	Interrupt Code	Input
HOLD	DMA Request	Input
HOLDA	Hold Acknowledge	Output
LOAD	Load Interrupt	Input
RESET	Reset	Input
V_{BB}, V_{CC}, V_{DD}, V_{SS}	Power and Ground reference	

Figure 18-2. TMS 9900 Signals and Pin Assignments

Pins A0 - A14 provide the 15-bit Address Bus. Note that Texas Instruments' literature numbers bits and pins from left to right; therefore, address line A0 represents the most significant address bit, where as address line A14 represents the least significant address bit.

D0 - D15 provide a 16-bit bidirectional Data Bus. Once again, D0 represents the most significant data bit in Texas Instruments' literature.

Remaining signals may be divided into bus control, interrupt control, and timing.

External logic must provide four clock signals, Φ1, Φ2, Φ3, and Φ4. These are provided by the TIM 9904, described later in this chapter.

Any memory access operation begins with an address being output via the Address Bus. **The TMS 9900 CPU identifies a stable address on the Address Bus by outputting $\overline{\text{MEMEN}}$ low.**

If the memory access operation is an instruction fetch, the IAQ is output high.

If the memory access is a read, then the TMS 9900 outputs a high level via DBIN. Memory interface logic must interpret the high DBIN level as a signal to place data on the Data Bus.

If the memory access is a memory write, then the TMS 9900 CPU outputs a low pulse via $\overline{\text{WE}}$. Memory interface logic must use the low $\overline{\text{WE}}$ pulse to signal that valid data is on the Data Bus, and to store it in the addressed memory location. $\overline{\text{WE}}$ low does not last as long as DBIN high.

When external logic cannot respond to a memory access in the available time, it requests a Wait state by inputting READY low. The CPU acknowledges by outputting WAIT high.

CRUCLK, CRUIN, and CRUOUT are three signals used to implement single-bit or serial data transfers via the CRU interface.

CRUOUT is used to output bits of data to the I/O devices, and CRUIN is used to retrieve input data from the I/O devices. CRUCLK is active during output operations only, and defines when data bits on CRUOUT are valid.

Let us now look at interrupt control signals.

There is a single interrupt request input, $\overline{\text{INTREQ}}$, which must be held low by any external device requesting an interrupt. External devices identify themselves via control signals IC0 - IC3. Thus, an interrupt request must be accompanied by the appropriate input at IC0 - IC3.

Observe that there is no interrupt acknowledge signal.

For DMA operations, external logic requests access to the System Bus by inputting $\overline{\text{HOLD}}$ low. The CPU acknowledges the Hold request by outputting HOLDA high.

$\overline{\text{LOAD}}$ is a nonmaskable interrupt.

$\overline{\text{RESET}}$ is a typical system Reset signal. However, TMS 9900 Reset logic uses the device's interrupt capabilities; therefore, we will describe the Reset operation in detail when discussing TMS 9900 interrupt capabilities in general.

TMS 9900 TIMING AND INSTRUCTION EXECUTION

TMS instructions execute as a sequence of machine cycles, each of which contains two clock periods. Clock periods are timed by four clock signals, Φ1, Φ2, Φ3, and Φ4, as illustrated in Figure 18-3. Note that Φ2 is the first phase of each clock period, and that Φ1 is the last phase.

The simplest instruction execution machine cycle is an internal operations cycle. No external bus signals are active during this machine cycle, and no memory or I/O access occurs. **Timing for an internal operations machine cycle will consist of two clock periods, as illustrated in Figure 18-3.**

TMS 9900 INTERNAL OPERATIONS MACHINE CYCLE

MEMORY ACCESS OPERATIONS

TMS 9900 memory access operations may consist of a memory read or a memory write. An instruction fetch is a minor variation of a memory read.

Figure 18-4 illustrates memory read machine cycle timing.

$\overline{\text{MEMEN}}$ goes low at the beginning of any memory access machine cycle and stays low for the entire machine cycle.

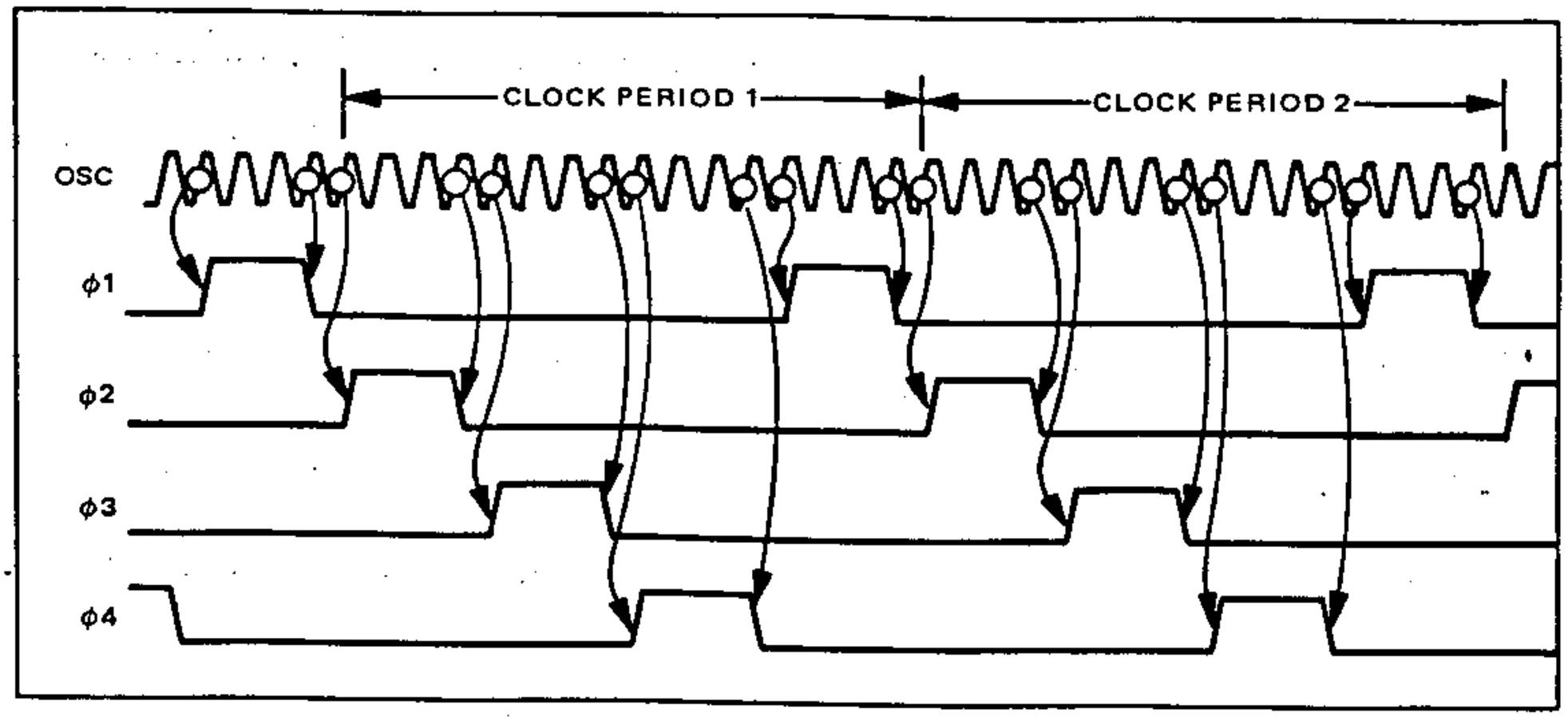


Figure 18-3. TMS 9900 Clock Periods and Timing Signals as Generated by the TIM 9904

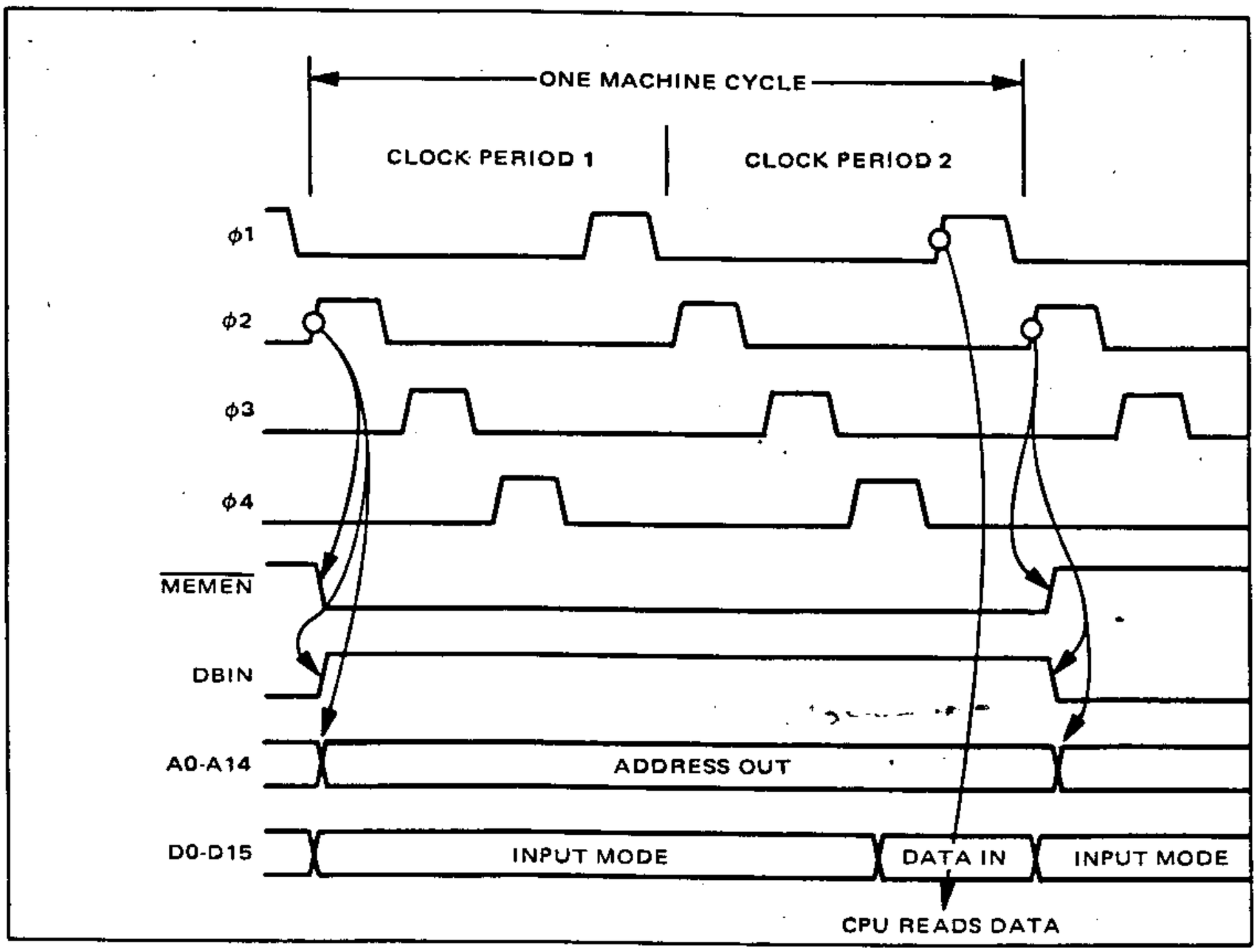


Figure 18-4. A TMS 9900 Memory Read Machine Cycle

DBIN goes high at the beginning of the memory read machine cycle and stays high for the entire machine cycle. External logic can therefore use $\overline{\text{MEMEN}}$ low as a memory address indicator while DBIN high identifies the read operation.

A memory address is output stable on the Address Bus for the entire machine cycle.

The Data Bus operations during a memory read machine cycle represent the only unusual characteristics of the machine cycle. Input data needs to be stable during the Φ1 high pulse of the second clock period. However, the Data Bus is connected to input logic for the entire memory read machine cycle and for a portion of the next machine cycle. Thus, during a memory read machine cycle, external logic cannot access the Data Bus to perform direct memory access, or any other operations, on the assumption that the Data Bus is free until Data In becomes stable. Moreover, since the Data Bus is held by data input logic of the CPU during the next machine cycle, a memory read machine cycle cannot be followed by a memory write machine cycle. **A memory read machine cycle must be followed by an internal operations machine cycle, or by another memory read machine cycle.**

The only difference between an instruction fetch machine cycle and a memory read machine cycle is the fact that **during an instruction fetch machine cycle, IAQ is output high,** along with DBIN, for the duration of the machine cycle.

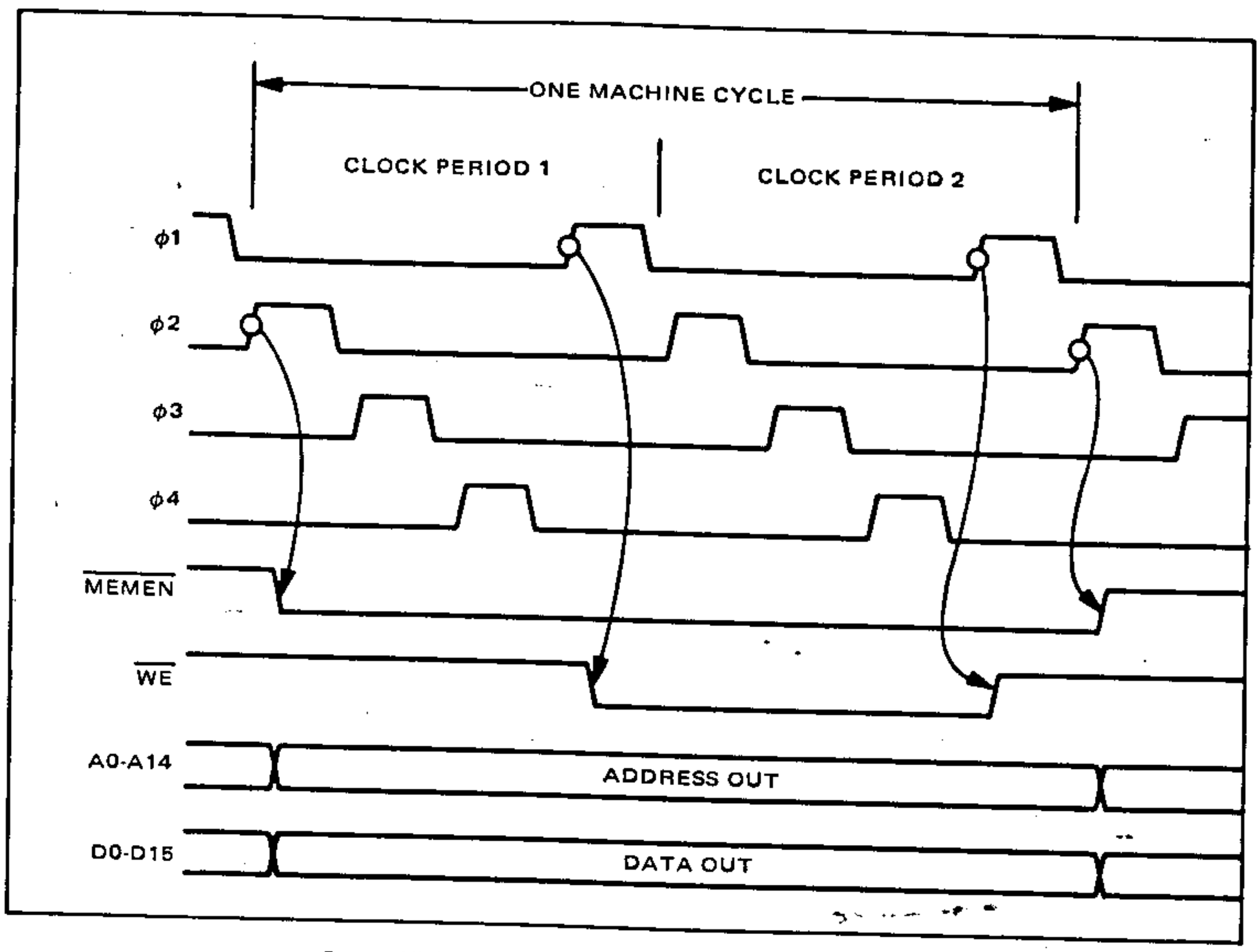
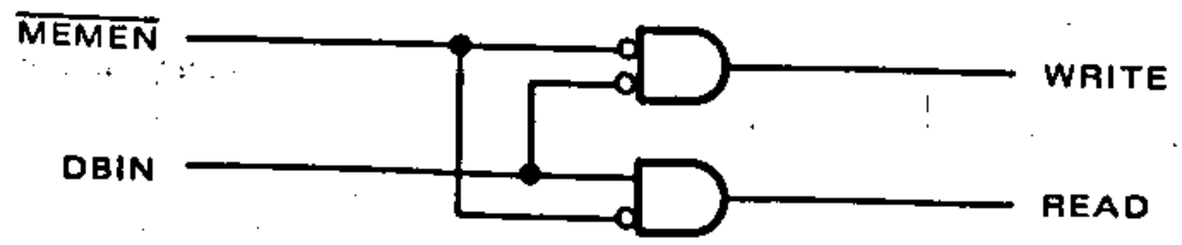


Figure 18-5. A TMS 9900 Memory Write Machine Cycle

Memory write machine cycle timing is illustrated in Figure 18-5. In this illustration, we see that data is output stable on the Data Bus for the entire duration of the memory write machine cycle. The Data Bus is not held by output logic beyond this single machine cycle. Thus, no restrictions are placed on the type of machine cycle which can follow a memory write machine cycle. Even though data output is stable for the entire memory write machine cycle, the write

enable strobe $\overline{WE}$ does not go low until close to the end of the first clock period. In many cases it is easier to use NOT DBIN as a write control signal. Here is the necessary logic:



TMS 9900 instruction execution machine cycle sequences are not always self-evident; therefore, let us look at some memory reference examples.

Memory address computations make machine cycle sequences quite complex, particularly for two-operand instructions. Fortunately, the exact machine cycle sequences are rarely of any consequence to you as a programmer or logic designer. The eventual number of machine cycles required to execute an instruction (and therefore its execution time) is important.

TMS 9900 INSTRUCTION EXECUTION SEQUENCES

Generally stated, **instruction execution proceeds as follows:**

1) The instruction object code is fetched.
2) The first operand address is computed.
3) The second operand address (if there is one) is computed.
4) Any operation that may be required is performed.
5) If a result is generated, it is returned to the second operand address.

Let us look at operand address computations using the ADD instruction (A) as a general example. First consider the instruction in its simplest form — where the contents of one register are added to the contents of another register:

A R1,R2

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	MEMORY READ	18-4	Fetch R1 contents
4	ALU	18-3	
5	MEMORY READ	18-4	Fetch R2 contents
6	ALU	18-3	Add R1 and R2 contents
7	MEMORY WRITE	18-5	Store sum in R2

Now consider the same instruction's execution, but using implied memory addressing for the first operand:

A *R1,R2

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	MEMORY READ	18-4	Fetch R1 contents
4	ALU	18-3	Use R1 contents as a memory address (implied addressing)
5	MEMORY READ	18-4	Fetch contents of implied address location
6	ALU	18-3	
7	MEMORY READ	18-4	Fetch R2 contents
8	ALU	18-3	Add data fetched in cycles 5 and 7
9	MEMORY WRITE	18-5	Store sum in R2

If the second (destination) operand uses direct addressing, here is the machine cycle sequence:

```
A     *R1,@LABEL
```

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	MEMORY READ	18-4	Fetch R1 contents
4	ALU	18-3	Use R1 contents as a memory address
5	MEMORY READ	18-4	Fetch contents of implied address location
6,7,8	ALU	18-3	
9	MEMORY READ	18-4	Fetch the second instruction object code word; it holds the direct address
10	ALU	18-3	
11	MEMORY READ	18-4	Fetch contents of directly addressed memory word
12	ALU	18-3	Add words fetched in cycles 5 and 11
13	MEMORY WRITE	18-5	Store sum in directly addressed memory word

Indexed, direct addressing results in the following sequence:

```
A     *R1,@LABEL(5)
```

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	MEMORY READ	18-4	Fetch R1 contents
4	ALU	18-3	Use R1 contents as a memory address
5	MEMORY READ	18-4	Fetch contents of implied address location
6	ALU	18-3	
7	MEMORY READ	18-4	Fetch the second instruction object code word; it holds the direct address
8	ALU	18-3	
9	MEMORY READ	18-4	Fetch R5, the Index register contents
10	ALU	18-3	Add direct address and index
11	MEMORY READ	18-4	Fetch contents of memory word addressed by cycle 10 addition
12	ALU	18-3	Add memory words fetched in cycles 5 and 11
13	MEMORY WRITE	18-5	Store sum in memory word addressed by cycle 10 addition

If the first operand-implied address specified an auto-increment, we must add one more machine cycle:

```
A     *R1+,@LABEL(5)
```

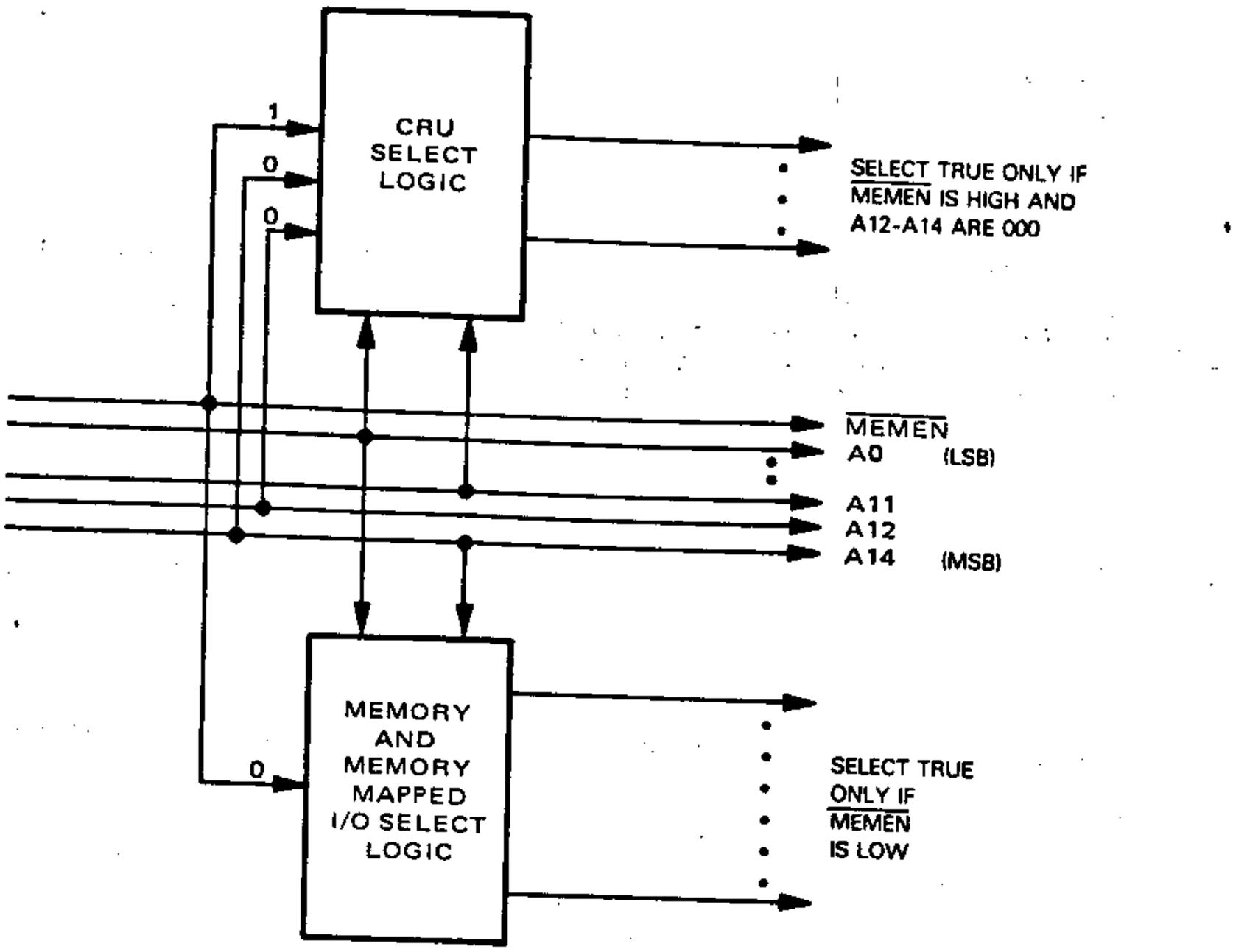
Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	MEMORY READ	18-4	Fetch R1 contents
4	ALU	18-3	Increment fetched R1 contents
5	MEMORY WRITE	18-5	Write incremented R1 contents back to R1
6	MEMORY READ	18-4	Fetch contents of implied address location
7	ALU	18-3	
8	MEMORY READ	18-4	Fetch the second instruction object code word; it holds the direct address
9	ALU	18-3	
10	MEMORY READ	18-4	Fetch R5, the Index register contents
11	ALU	18-3	Add direct address and index
12	MEMORY READ	18-4	Fetch contents of memory word addressed by cycle 11 addition
13	ALU	18-3	Add memory words fetched in cycles 5 and 12
14	MEMORY WRITE	18-5	Store sum in memory word addressed by cycle 11 addition

MEMORY SELECT LOGIC

$\overline{\text{MEMEN}}$ discriminates between memory and I/O accesses. It is therefore very important that $\overline{\text{MEMEN}}$ low be a necessary component for any memory select.

You can map I/O into the memory space of the TMS 9900. This is true of any microprocessor. Memory addresses that select I/O devices will, of course, also require $\overline{\text{MEMEN}}$ low as a contributor to I/O device select logic.

$\overline{\text{MEMEN}}$ as a contributor to select logic may be illustrated as follows:



The three high-order address lines. A12. A13. and A14. are not used to address CRU bits. When addressing a CRU bit. these lines are all low. They are not low during execution of externally defined I/O instructions; therefore. A12. A13. and A14 low must be a prerequisite for any CRU bit select.

TMS 9900 I/O INSTRUCTION TIMING

All TMS 9900 I/O instructions transfer serial data via the Communication Register Unit (CRU). (This excludes I/O which is addressed as TMS 9900 memory space.)

There are four types of TMS 9900 I/O instructions. They are:

1) **Data input.** Anywhere from 1 to 16 bits of data may be transferred from the CRU bit field to memory
2) **Data output.** This is the simple reverse of data input. Anywhere from 1 to 16 bits of data may be output from memory to the CRU bit field.
3) **Bit test.** Any bit in the CRU bit field may be tested. The tested bit is input and stored in the Equal bit of the Status register. Thence. condition branch instructions can be used to test the bit level.
4) **Externally defined I/O instructions.** These instructions generate I/O control signals. but they transfer no data.

Timing for CRU output and input machine cycles is illustrated in Figures 18-6 and 18-7, respectively. Each of these figures shows two bits of data being transferred. (You should not attach any special significance to this fact: depending on the instruction being executed. anywhere from 1 to 16 bits may be transferred.) CRU machine cycles are executed contiguously. one per bit.

Every CRU I/O instruction will require a memory reference machine cycle, together with one or more CRU machine cycles. For example, **when an STCR instruction is executed to input data from the CRU to the CPU, the following machine cycle sequence will occur:**

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch Instruction Code
2	ALU	18-3	Decode Instruction
a Cycles, where $0 \leq a \leq 4$			Obtain Destination Address
3 + a	MEMORY READ	18-4	Fetch Destination Memory Word Contents
4 + a	ALU	18-3	
5 + a	MEMORY READ	18-4	Fetch R12
6 + a 7 + a	ALU ALU	18-3	Compute CRU Starting Address and Prepare Control Signals
i Cycles	CRU IN	18-7	Input i CRU Bits
8 + a + i 9 + a + i	ALU ALU	18-3	Load CRU Bits in Temporary Register
r Cycles			Fill Upper Bits of Byte or Word With Zeroes If $i > 8$, $r = 15 - i$; if $i \leq 8$, $r = 7 - i$
10 + a + i + r to 12 + a + i + r	ALU	18-3	Prepare to Store Memory Word
13 + a + i + r	MEMORY WRITE	18-5	Output Assembled Word to Memory Location Whose Contents Were Fetched in Machine Cycle 3 + a

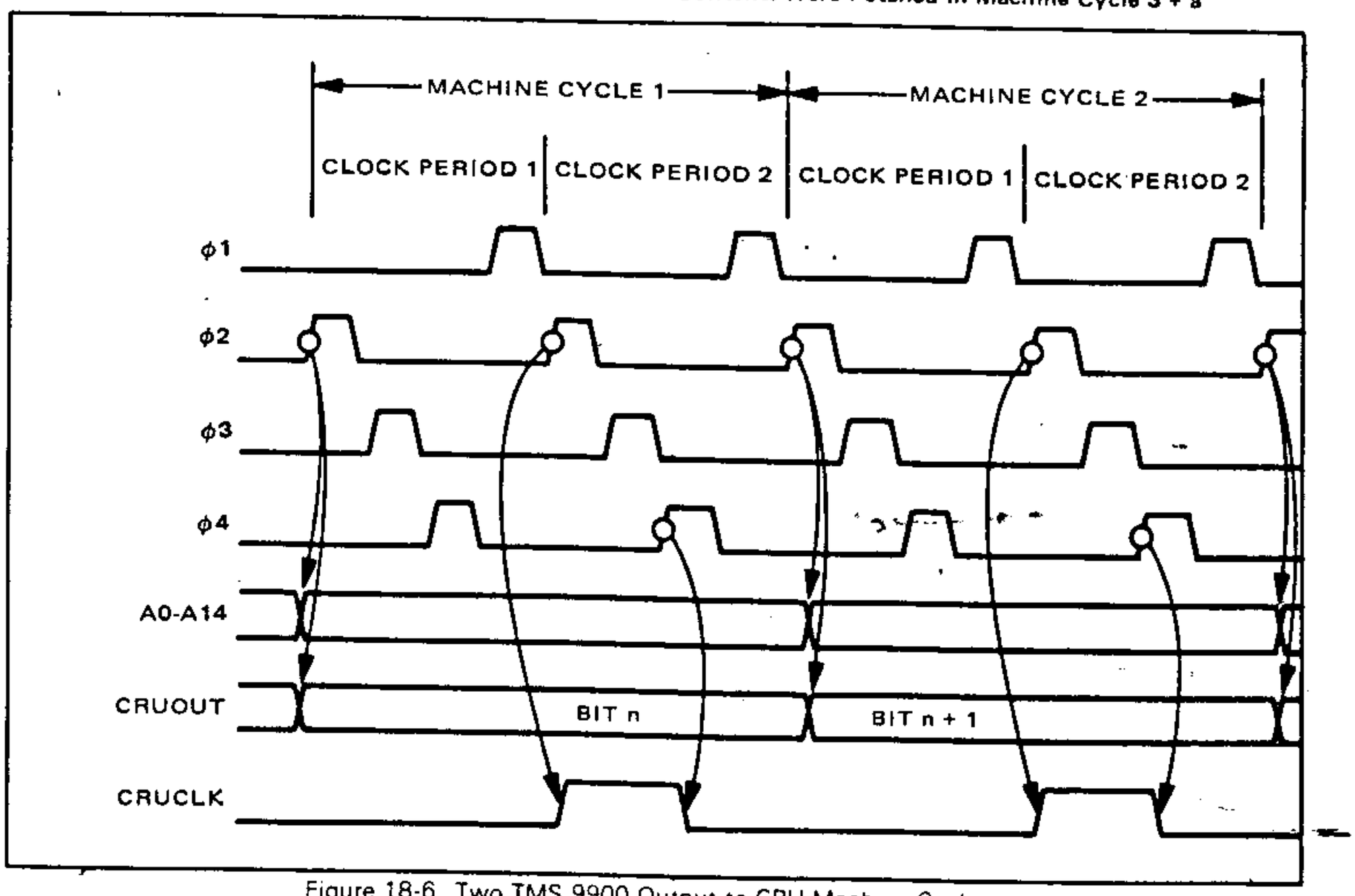


Figure 18-6. Two TMS 9900 Output-to-CRU Machine Cycles

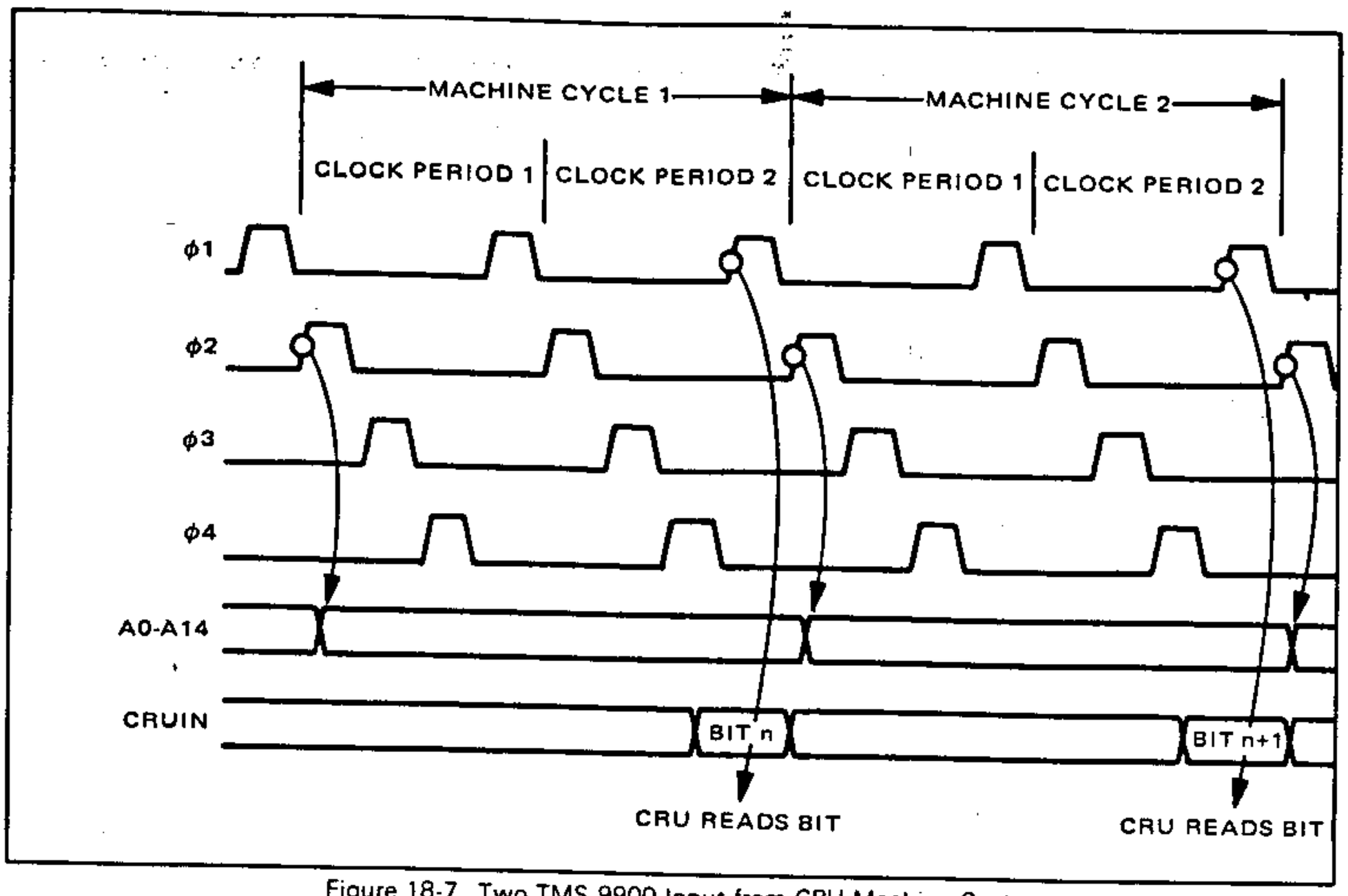


Figure 18-7. Two TMS 9900 Input-from-CRU Machine Cycles

An LDCR instruction outputs a sequence of 1 to 16 data bits to a CRU bit field. Here is the LDCR instruction machine cycle sequence:

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
a Cycles where $0 \leq a \leq 4$			Obtain source address
3+a	MEMORY READ	18-4	Fetch source memory word contents
4+a to 7+a	ALU	18-3	Prepare for data transmission
8+a	MEMORY READ	18-4	Fetch R12
9+a	ALU	18-3	Compute CRU starting address
i Cycles	CRU OUT	18-6	Output i bits to CRU
10+a+i	ALU	18-3	Machine cycle to conclude instruction

The SBO and SBZ instructions set or reset an addressed CRU bit; in essence, these instructions output one data bit. Here is the machine cycle sequence via which the bit output occurs:

Cycle	Type	Figure	Function
1	MEMORY READ	18-4	Fetch instruction object code
2	ALU	18-3	Decode instruction
3	ALU	18-3	Decode instruction
4	MEMORY READ	18-4	Fetch R12
5	ALU	18-3	Compute CRU address
6	CRU OUT	18-6	Output to addressed CRU bit

The TB instruction inputs one CRU bit; its timing is identical to the SBO and SBZ instructions, except that machine cycle 6 is a CRU IN machine cycle.

The Address Bus is used in an unusual way during a CRU machine cycle. As we have already stated, the CRU bit field is 4096 bits wide — addressed by 12 of the 15 Address Bus lines. **The three high-order Address Bus lines are used to identify I/O control instructions, as defined in Table 18-1.** We can conclude from Table 18-1 that when $\overline{\text{MEMEN}}$ is high and the three high-order Address Bus lines are all low, an I/O transfer is occurring. Otherwise, one of five externally defined I/O control instructions is being executed. There are dedicated functions for these five I/O controls in TM 990 minicomputer systems; these are shown in Table 18-1. But to anyone who is simply building a microcomputer system around a TMS 9900, these five I/O states are undefined. **Thus, Figure 18-8 illustrates TMS 9900 systems' bus utilization during both CRU operations and externally defined I/O operations. If CRU SEL and MEMEN are high, CRU Select logic will be active.**

Table 18-1. High-Order Address Bus Line Used by TMS 9900 I/O Instructions

INSTRUCTION MNEMONIC	INSTRUCTION TYPE	(MSB) A14	A13	A12	FUNCTION
LDCR	Output	0	0	0	Output data to CRU
SBO	Output	0	0	0	Set CRU bit to 1
SBZ	Output	0	0	0	Reset CRU bit to 0
STCR	Input	0	0	0	Input data from CRU
TB	Test (Input)	0	0	0	Input CRU bit to Equal status bit
IDLE	Control	0	1	0	Enter HALT condition
RSET	Control	0	1	1	Reset the Interrupt mask
CKOF	Control	1	0	1	Real time clock on } These are TM 990 uses. Instructions are undefined in a TMS 9900 system.
CKON	Control	1	1	0	Real time clock off
LREX	Control	1	1	1	Execute bootstrap

Externally defined instructions output 0 on the 12 low-order Address Bus lines, A0 - A11; in addition, CRUCLK pulses are output as part of the instruction executions.

CRUCLK is an active CRU output strobe only. This signal pulses high whenever a valid level is present on the CRUOUT signal line. **There is no pulse for CRUIN.** External logic must generate its own strobe if it is needed, by combining $\overline{\text{MEMEN}}$ high with a valid bit pattern on the Address Bus.

CRU instructions that test the level of a bit are, to external logic, no different from CRU input instructions. External logic is required to return, via CRUIN the level of the selected bit. The fact that the CPU interprets this input as status, rather than data, is immaterial to external logic.

THE WAIT STATE

Additional Wait State clock periods may be inserted between clock periods 1 and 2 of any memory access machine cycle. Timing is illustrated in Figure 18-9. At the rising edge of Φ1 of clock period 1, the CPU samples the READY input signal. If this signal is low, then the next clock period is a Wait clock period. During a Wait cycle, the WAIT output signal is high; all other output signals hold the levels they had during clock period 1.

A Wait State can last for any number of clock periods. During the Φ1 high pulse of every Wait clock period, the CPU samples the level of the READY input. As soon as READY is sampled high, the Wait State ends. The next clock period becomes clock period 2 of the machine cycle, and the memory operation is completed.

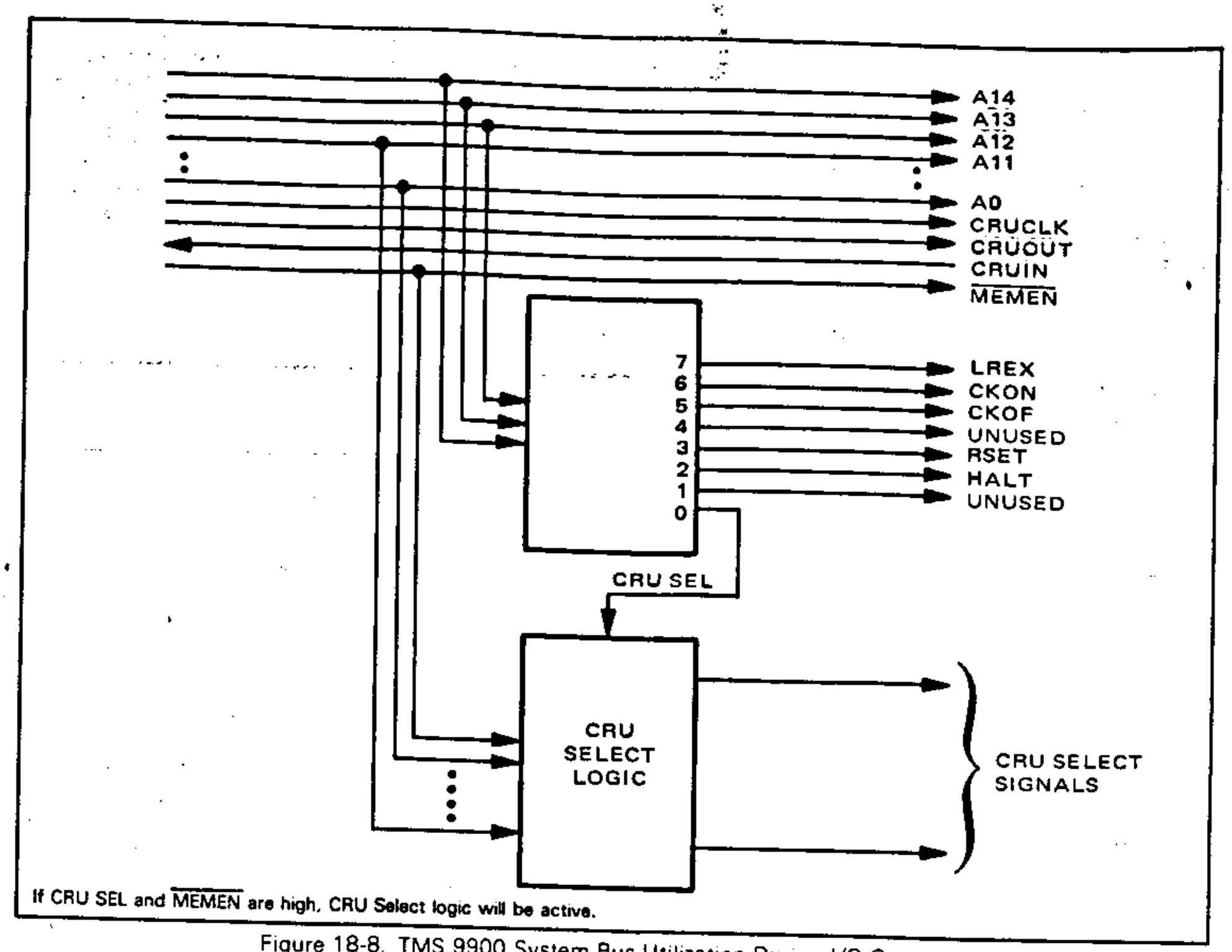


Figure 18-8. TMS 9900 System Bus Utilization During I/O Operations

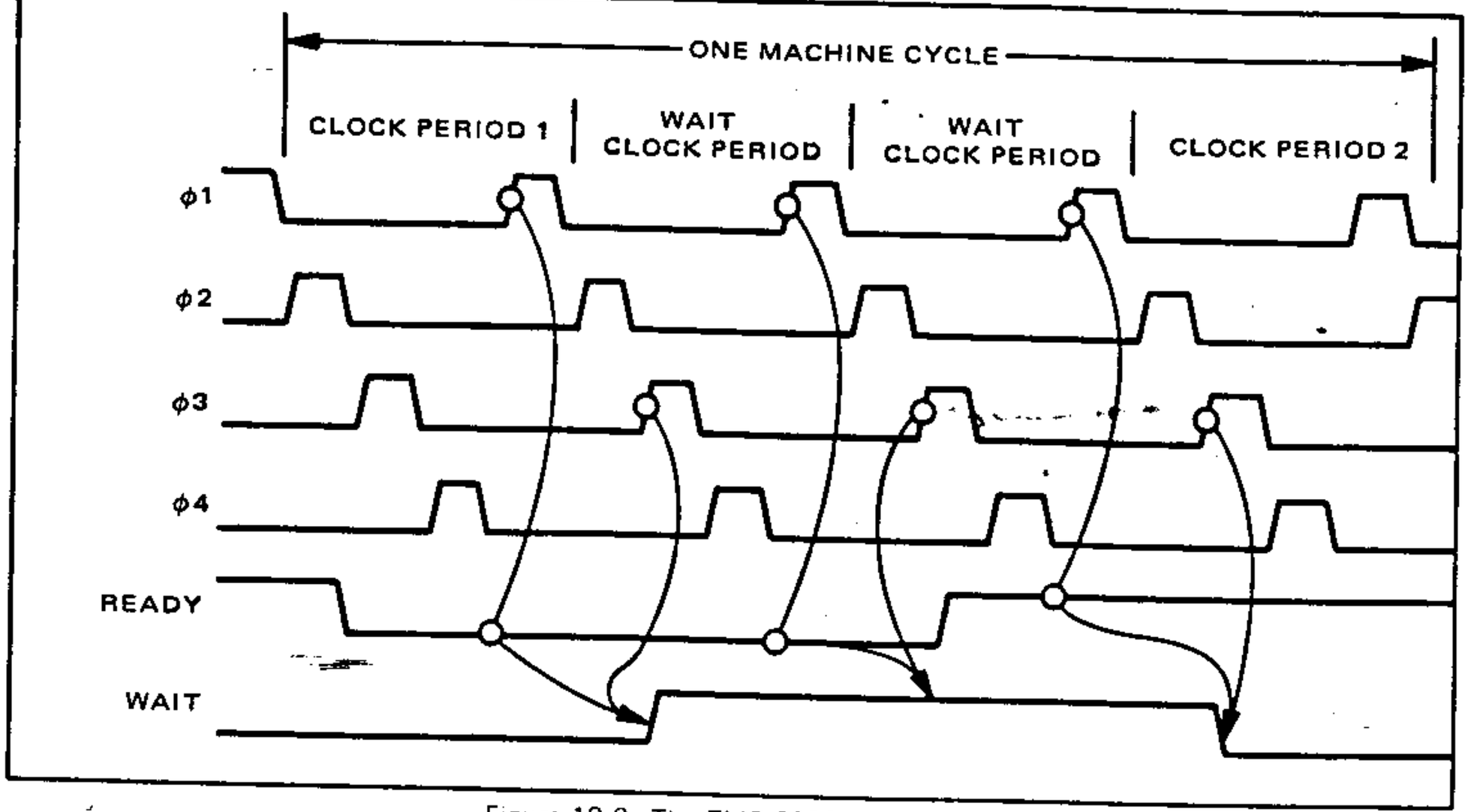


Figure 18-9. The TMS 9900 Wait State

THE HOLD STATE

The TMS 9900 has a typical microcomputer Hold State, used to enable direct memory access operations. External logic initiates a Hold State by inputting $\overline{\text{HOLD}}$ low. At the beginning of the next nonmemory reference machine cycle. the CPU floats its Address and Data Busses. together with the DBIN. $\overline{\text{MEMEN}}$ and $\overline{\text{WE}}$ control signals. HOLDA is output high as a Hold Acknowledge. **Timing is illustrated in Figure 18-10.**

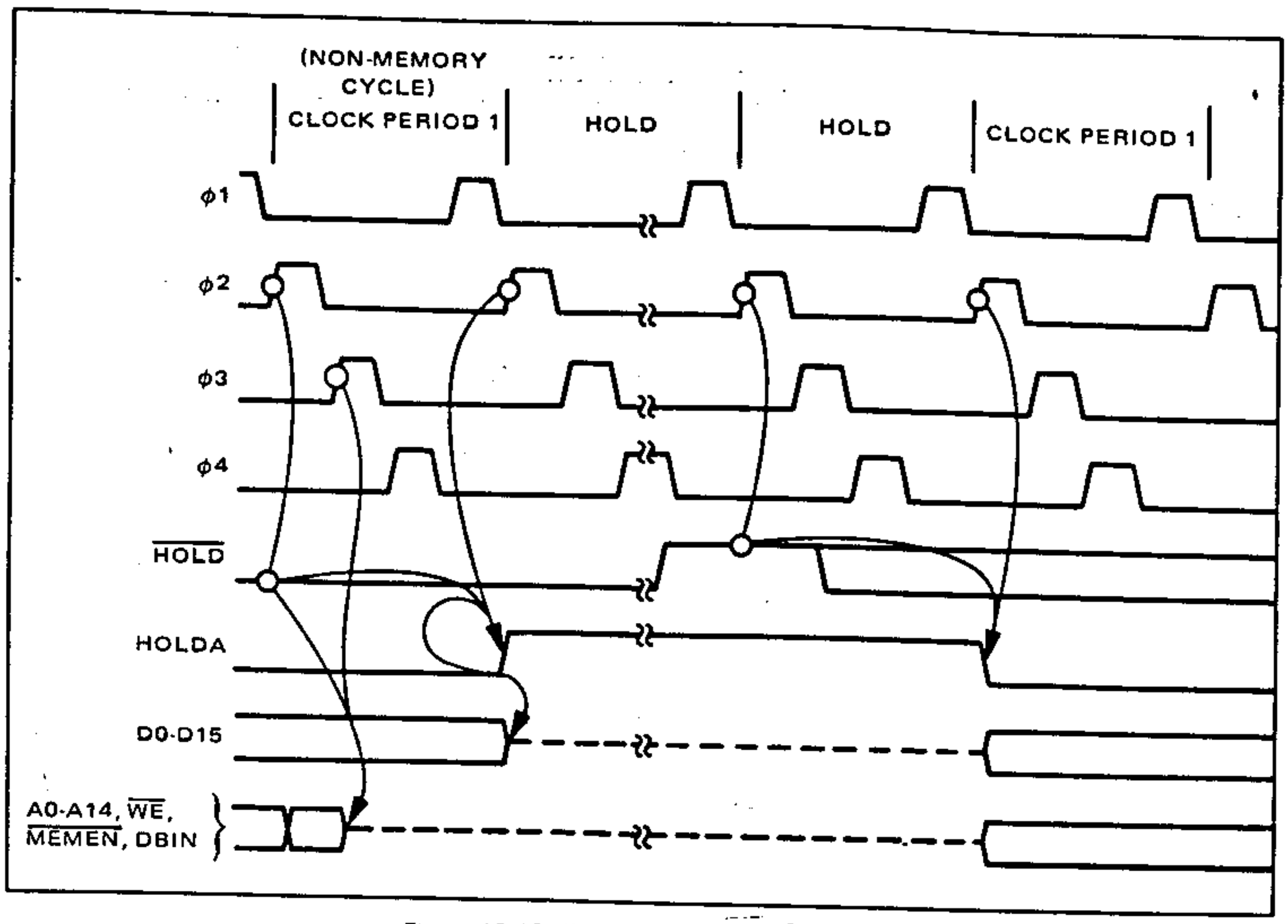


Figure 18-10. TMS 9900 Hold State Timing

The Hold State lasts until external logic raises $\overline{\text{HOLD}}$ high again.

It is up to external logic to perform all operations associated with a DMA transfer. The CPU simply floats the System Bus in response to a Hold request. As soon as the TMS 9911 device is available, this will be the part of choice to use in all TMS 9900 microcomputer systems that use direct memory access logic. Any of the other DMA devices described in Volume 3 may also be used.

The only nonobvious aspect of Figure 18-10 is the fact that Data Bus timing, during normal instruction execution, differs from other System Bus signal timing. Figure 18-10 highlights this fact by showing the Data Bus floating at the beginning of the first HOLD clock period. while other signals float earlier in the preceding clock period. This is not a particularly significant event. The entire System Bus is floating once the HOLD clock period has begun. However. the actual tristate condition for any signal begins at that point in the preceding clock period when the signal is no longer being driven by current operations.

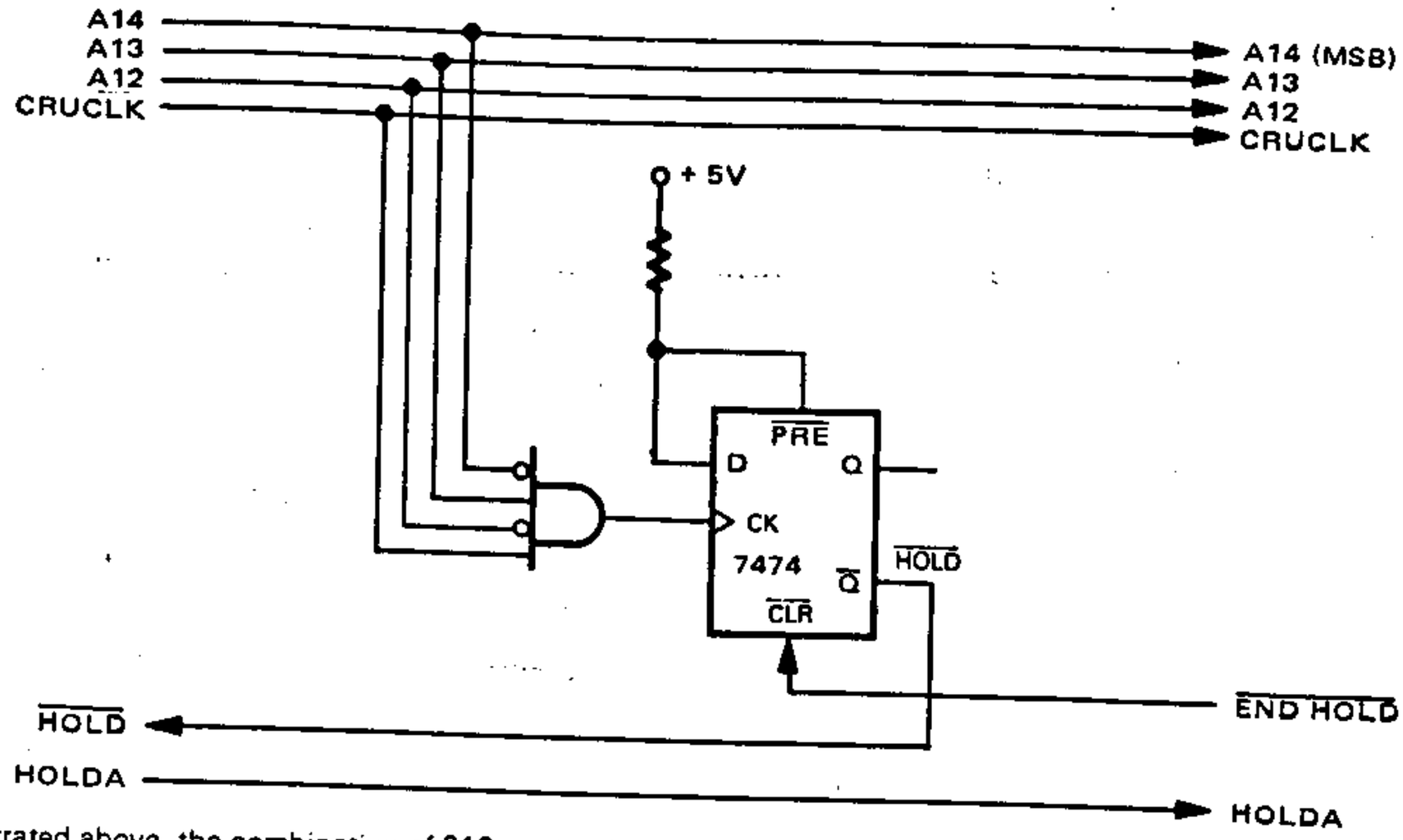
THE HALT STATE

The TMS 9900 IDLE I/O instruction generates a Halt State. When this instruction is executed. the CPU suspends all program execution and internal operations. You must terminate the Idle condition with an interrupt request or a low $\overline{\text{LOAD}}$ or $\overline{\text{RESET}}$ input. ($\overline{\text{LOAD}}$ and $\overline{\text{RESET}}$ are treated as interrupts as we will describe soon.)

The TMS 9900 CPU does not relinquish the System Bus while halted. That is to say. after an IDLE instruction has been executed. no System Bus lines are floated.

The IDLE instruction is usually executed when program logic requires that the CPU wait for an interrupt. or when external logic is computing a real-time interval — which will be terminated with an interrupt request.

You can, if you wish, initiate a DMA transfer by executing an IDLE instruction. In order to do this, you must create a HOLD request from the Address Bus output characteristic of the IDLE instruction's execution. This may be illustrated as follows:



As illustrated above. the combination of 010 on the three high-order Address Bus lines. along with the CRUCLK pulse, identifies the IDLE instruction. Since the process of floating the System Bus will remove the conditions which generated a Hold request. these conditions are used to clock a flip-flop. Thus. external logic which receives the Hold acknowledge signal and takes control of the System Bus must subsequently reset the Hold request flip-flop in order to remove the Hold condition. That is to say. **program logic can begin a Hold state within a Halt state, but it cannot end this combination. Two steps are needed to terminate a Hold within a Halt. The Hold request must be removed, then an interrupt request must follow to terminate the Halt.**

TMS 9900 INTERRUPT PROCESSING LOGIC

The TMS 9900 has complex and capable interrupt processing logic. Sixteen levels of external interrupt are available. Sixteen software interrupts are also available. Fifteen of the sixteen external interrupts are maskable: the nonmaskable interrupt has highest priority and is the system Reset interrupt. There is. in addition. a non-maskable Load interrupt. External interrupts may be summarized as follows:

$\overline{\text{LOAD}}$ / $\overline{\text{RESET}}$	Priority	0	Non-maskable, Equal Highest Priority Interrupts
Maskable Levels of External Interrupt	Priority	1	
	Priority	2	
	Priority	3	
	Priority	4	
	Priority	5	
	Priority	6	
	Priority	7	
	Priority	8	
	Priority	9	
	Priority	10	
	Priority	11	
	Priority	12	
	Priority	13	
	Priority	14	
	Priority	15	Lowest Priority Interrupt

External logic identifies the priority of its interrupt request via the IC0, IC1, IC2, and IC3 inputs, as follows:

IC0	IC1	IC2	IC3	Priority
0	0	0	0	Should not be input by external logic - highest external
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15 lowest external

Software interrupts are executed via the XOP instruction. There are, in addition, instructions that parallel the $\overline{\text{RESET}}$ and $\overline{\text{LOAD}}$ interrupts. We will describe these instructions in due course.

TMS 9900 INTERRUPT VECTOR MAP

Each one of the external interrupts has two dedicated memory words via which vectoring is enabled following an interrupt acknowledge. Figure 18-11 illustrates the memory map associated with interrupt vectoring. The memory addresses in Figure 18-11 are byte addresses as seen by the programmer. Remember, the low-order bit of the address shown in Figure 18-11 is not output on the Address Bus; therefore, you must divide the memory addresses shown in Figure 18-11 by 2 in order to generate the address which will be seen by external memory.

The memory words dedicated to interrupt vectoring, as illustrated in Figure 18-11, can be read-only memory, read/write memory, or any combination of the two. Obviously, read-only memory will be used in applications that have dedicated interrupt service routines for specific interrupt requests. Read/write memory might be used in minicomputer-type applications where the interrupt response will depend on the application being serviced.

Interrupt masking and priorities apply only to external interrupt requests. Interrupt masking priorities cannot be applied to software interrupts (the XOP instruction). Since program logic must generate the software interrupt, program logic can equally be relied on to know which software interrupt is to be executed, and whether the software interrupt is allowed by current program logic. That is to say, from the programmer's viewpoint, a software interrupt is simply the consequence of an XOP instruction's execution; you, as a programmer, can include an XOP instruction anywhere in a program, within or outside an interrupt service routine. XOP instructions might be used in response to error conditions, or to call any frequently used subroutines.

Let us begin by looking at the way in which external interrupts are processed.

Any external device wishing to request an interrupt must pull the $\overline{\text{INTREQ}}$ input low while simultaneously placing a 4-bit code at the IC0 - IC3 inputs. The CPU will acknowledge the interrupt, provided that its priority, as identified by the IC0 - IC3 inputs, is enabled. The interrupt will be acknowledged at the conclusion of the currently executing instruction. The BLWP and XOP instructions are exceptions; for the integrity of program logic, they demand that the next sequential instruction be executed. Therefore, if an interrupt request occurs while either of these two instructions is being executed, the interrupt will not be acknowledged until this instruction and the next instruction have been executed.

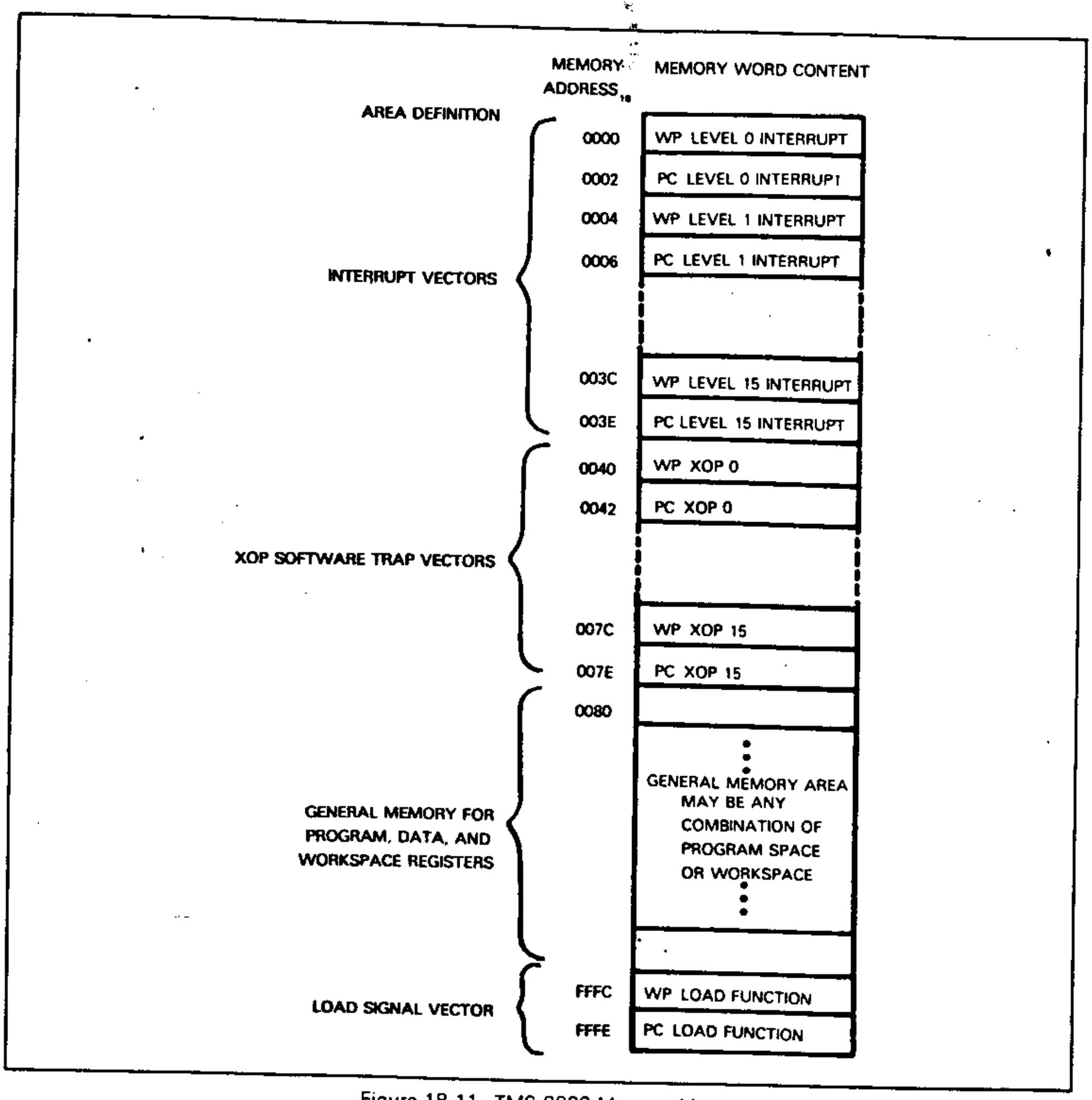


Figure 18-11. TMS 9900 Memory Map

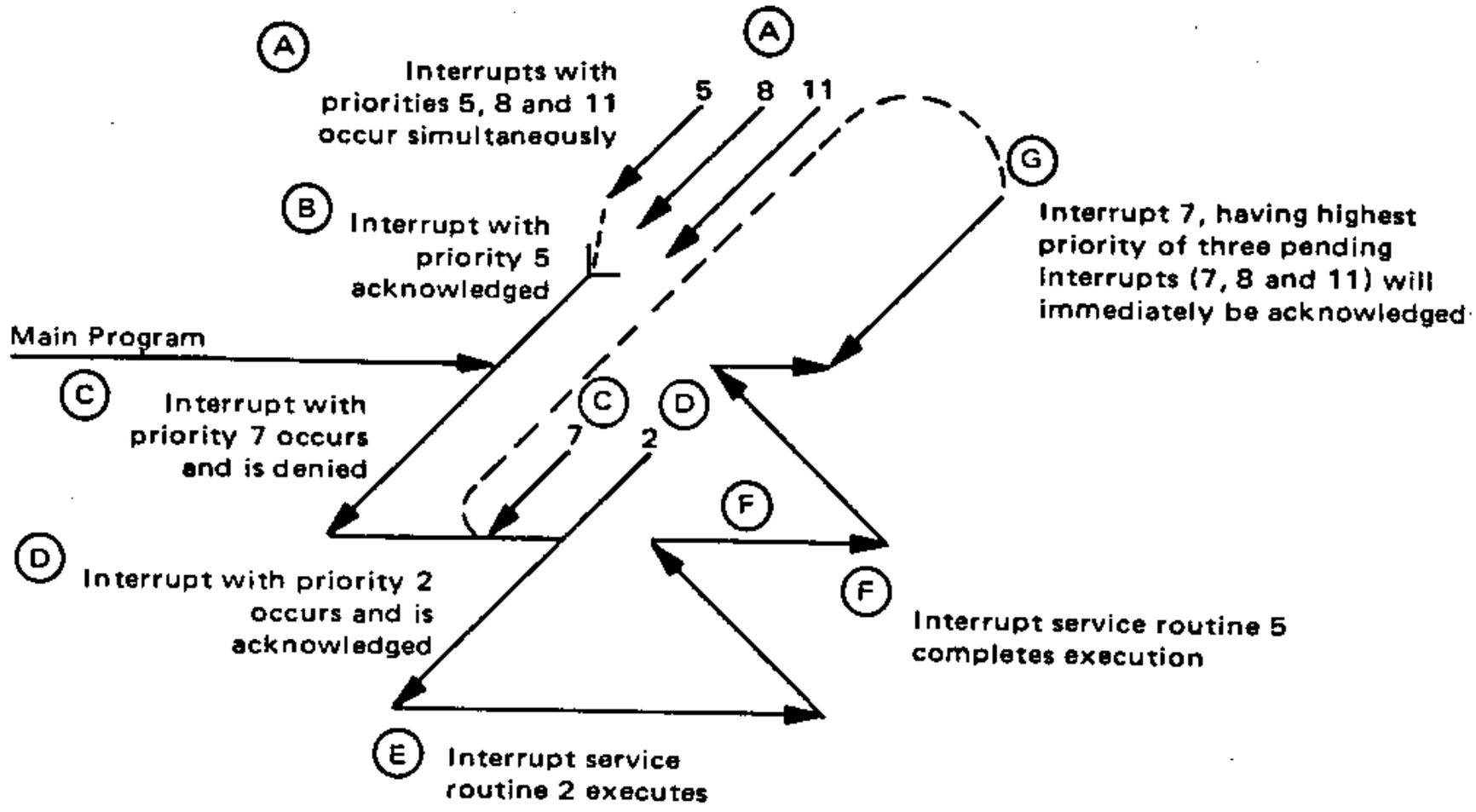
When an interrupt is acknowledged, the following machine cycles are executed:

Cycle	Type	Figure	Function
1	ALU	18-3	
2	MEMORY READ	18-4	Move new WP register contents from vector word to temporary storage
3	ALU	18-3	
4	MEMORY WRITE	18-5	Store status in new R15
5	ALU	18-3	Store IC0 - IC3 levels in four low-order Status bits
6	MEMORY WRITE	18-5	Store incremented PC in new R14
7	ALU	18-3	
8	MEMORY WRITE	18-5	Store old WP register contents in new R13
9	ALU	18-3	
10	MEMORY READ	18-4	Fetch new PC contents from vector word
11	ALU	18-3	Fetch new WP contents from temporary storage

Vector words are illustrated in Figure 18-11.

TMS 9900 NESTED INTERRUPT PRIORITIES

At the conclusion of the interrupt acknowledge sequence listed above, the priority of the acknowledged interrupt request, less one, is recorded in the four low-order Status register bits. Thus, subsequent interrupt requests will be acknowledged only if their priority is higher than that of the interrupt being serviced. That is to say, whenever an interrupt request occurs, CPU logic compares the levels input at IC0 - IC3 with the levels present in the four low-order Status register bits. If IC0 - IC3 is not greater than the mask, then the interrupt request will be acknowledged. If IC0 - IC3 is higher, then the interrupt request will not be acknowledged. Thus, **in the normal course of events, TMS 9900 interrupt priority logic disables all interrupts of equal or lower priority than an acknowledged interrupt,** while leaving higher priority interrupts enabled. **Priorities are maintained for the duration of the interrupt service routine.** This is illustrated in the following figure, which you should read in the sequence Ⓐ - Ⓑ - Ⓒ - Ⓓ - Ⓔ - Ⓕ - Ⓖ :



The interrupt priority arbitration logic of the TMS 9900 is exceptional among microcomputers. Most microcomputers arbitrate priorities at the instant interrupts are being acknowledged, and once an interrupt has been acknowledged, all interrupts are disabled. That is to say, interrupt priorities apply only during the acknowledge process. In contrast, the TMS 9900 maintains interrupt priorities for the duration of the interrupt service routine, as illustrated above.

The net effect of the interrupt response steps illustrated above is to perform a context switch while disabling all interrupts that have the same priority as the acknowledged interrupt, or that have a lower priority.

There are some very important and nonobvious advantages to initiating an interrupt service routine with a context switch.

Since the 16 new memory locations that will be used as general purpose registers may lie anywhere in read/write memory, you can store parameters that will be used by the interrupt service routine, in advance of the interrupt, in those memory locations that are ultimately to serve as general purpose registers for the duration of the interrupt service routine.

You can, if you wish, modify the interrupt priority scheme that will control nested interrupts. As we have already stated, if you do nothing about interrupt priorities, then any interrupt service routine may be interrupted by a higher priority external interrupt, but not by an external interrupt that has the same priority or a lower priority.

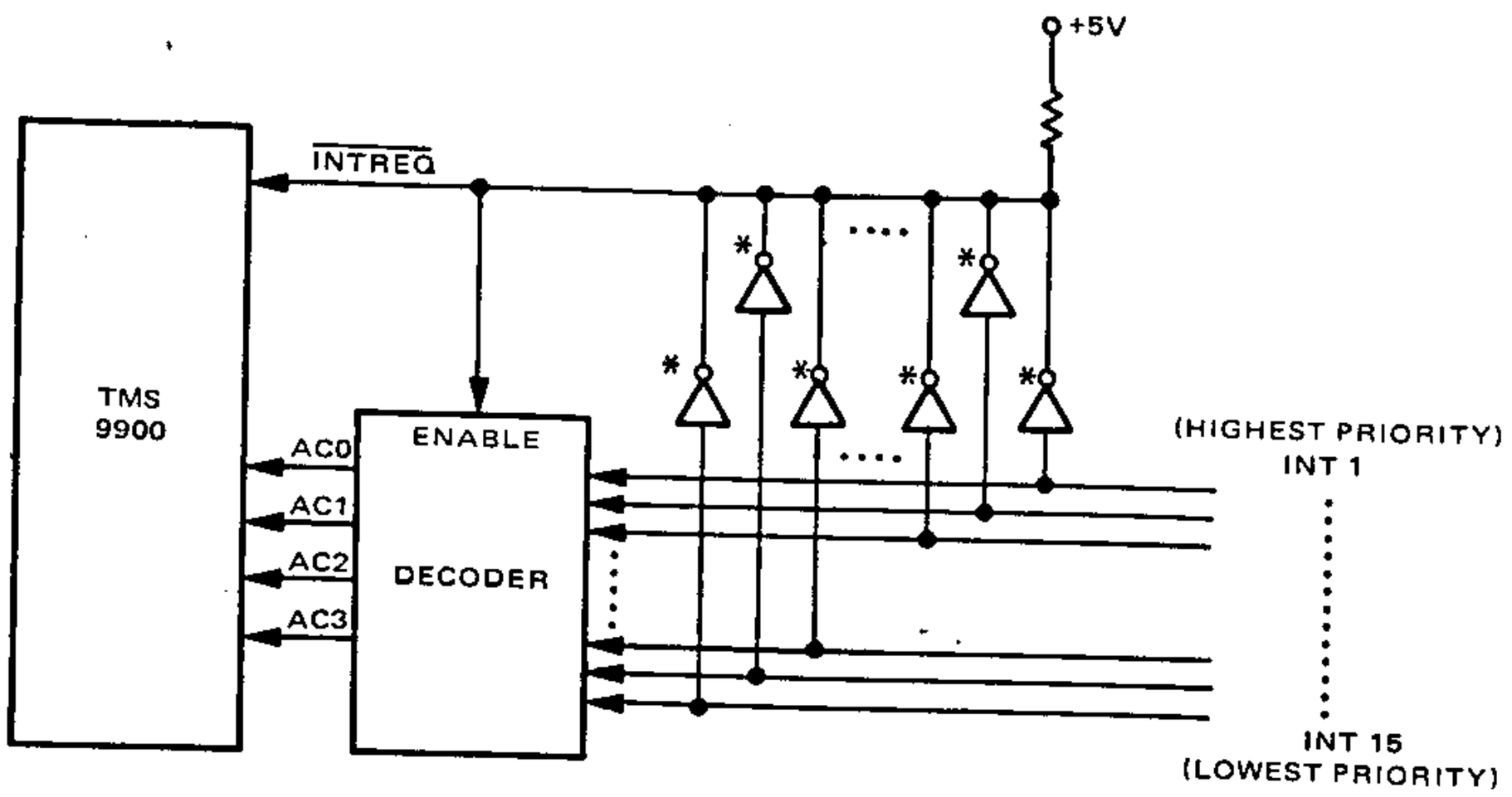
If you wish to eliminate nested interrupts entirely, then the first instruction executed within an interrupt service routine must be an LIMI 0 instruction (Load Interrupt Mask Immediate), which clears the four low-order Status register bits, thus disabling all maskable interrupts. A $\overline{\text{RESET}}$ or $\overline{\text{LOAD}}$ interrupt — or a level 0 external interrupt request — will still be acknowledged; these should be alarm conditions and not part of the normal interrupt logic of any microcomputer. You can execute variations of the LIMI instruction to increase or decrease the levels of priority that will be masked for the duration of any interrupt service routine (or for that matter, any subsequent instruction within the interrupt service routine) can load appropriate data into the four low-order bits of the Status register, thus changing the priority level at which all subsequent interrupt requests will be disabled.

All interrupt service routines should end with an RTWP (Return Workspace Pointer) instruction. The RTWP instruction performs a reverse context switch. which puts the central processing unit back to the logical environment which was interrupted. Observe that since the Status register is also saved during a forward context switch. the return instruction will restore whatever level of interrupt priorities existed at the instant the interrupt was acknowledged. You can. of course. modify the contents of General Purpose Registers R13. R14. and R15 in the course of an interrupt service routine's execution. This allows program logic to alter the conditions that will be restored when the return instruction executes a reverse context switch.

The TMS 9901 PSI, which we describe later in this chapter, provides multiple interrupt handling for TMS 9900 series CPUs. If your system does not include a TMS 9901, then external hardware required to support multiple interrupts in a TMS 9900 microcomputer system will not be as straightforward as the software response.

TMS 9900 MULTIPLE INTERRUPT HARDWARE CONSIDERATIONS

First of all. we must cope with the fact that if more than one interrupt request occurs simultaneously. then there will be competition on the $\overline{\text{INTREQ}}$ input. but there will also be competition at the four priority inputs. IC0 - IC3. Resolving competition on the $\overline{\text{INTREQ}}$ input is no problem; you can wire-OR interrupt requests from many devices to create the CPU input. But your external logic must make sure that only the highest priority combination of IC0 - IC3 appears at the TMS 9900 inputs. One method of doing this is to use latched decoders that create a 4-bit output corresponding to the highest level input. provided that the decoder is enabled by a latching signal. This may be illustrated functionally as follows:



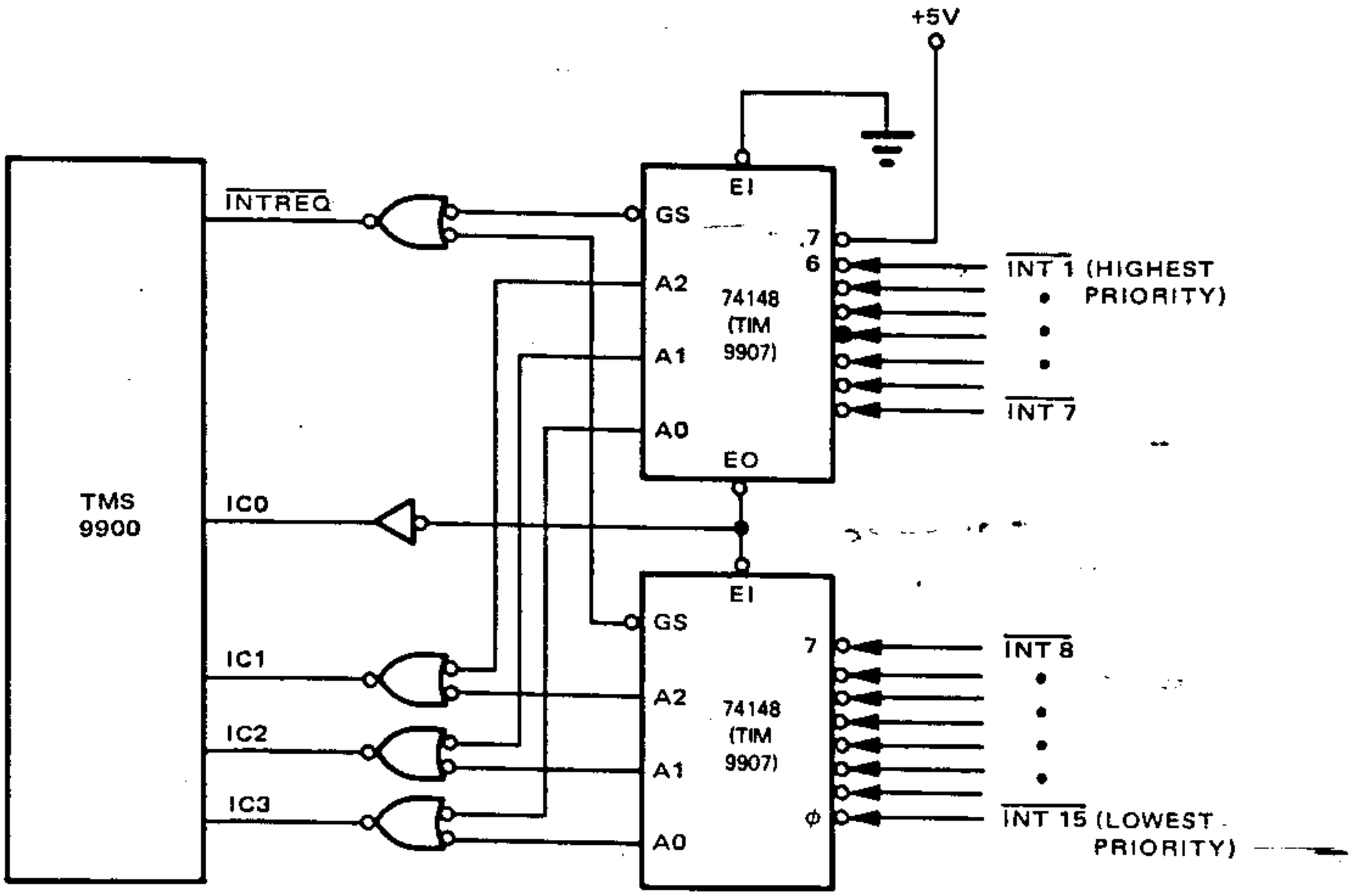
In the illustration above. 15 external interrupt requests are input to a decoder. These interrupt requests are high true. The 15 interrupt requests are buffered. inverted. and wire-ORed to create the master interrupt request $\overline{\text{INTREQ}}$. which is input to the CPU. This master interrupt request also enables the decoder. That is to say. when the enable input to the

decoder is high. the four outputs. IC0 - IC3 will be low. When the enable input to the decoder is low. IC0 - IC3 will output a 4-bit value as follows:

IC0	IC1	IC2	IC3	INT 1	INT 2	INT 3	INT 4	INT 5	INT 6	INT 7	INT 8	INT 9	INT 10	INT 11	INT 12	INT 13	INT 14	INT 15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0	0	1	0	0	1	*	*	*	*	*	*	*	*	*	*	*	*	*
0	0	1	1	0	0	1	*	*	*	*	*	*	*	*	*	*	*	*
0	1	0	0	0	0	0	1	*	*	*	*	*	*	*	*	*	*	*
0	1	0	1	0	0	0	0	1	*	*	*	*	*	*	*	*	*	*
0	1	1	0	0	0	0	0	0	1	*	*	*	*	*	*	*	*	*
0	1	1	1	0	0	0	0	0	0	1	*	*	*	*	*	*	*	*
1	0	0	0	0	0	0	0	0	0	0	1	*	*	*	*	*	*	*
1	0	0	1	0	0	0	0	0	0	0	0	1	*	*	*	*	*	*
1	0	1	0	0	0	0	0	0	0	0	0	0	1	*	*	*	*	*
1	0	1	1	0	0	0	0	0	0	0	0	0	0	1	*	*	*	*
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	*	*	*
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	*	*
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	*
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

* REPRESENTS A "DON'T CARE" BIT

If you do not use the TMS 9901. Texas Instruments suggests the following circuit to accomplish priority encoding:



External logic must maintain its interrupt request until it receives its own specific interrupt acknowledge. This need is obvious, since an interrupt request may be denied for a long time while higher priority interrupts are being serviced.

The problem is that the TMS 9900 has no interrupt acknowledge signals.

Interrupt acknowledge signals can be generated in one of two ways:

1) By using CRU bit instructions to set and reset external flip-flops that create interrupt acknowledge signals.
2) By decoding appropriate addresses on the Address Bus.

Figure 18-12 illustrates two possible configurations that will allow CRU bit set and reset instructions to generate interrupt acknowledge signals. The logic in Figure 18-12A generates a short interrupt acknowledge pulse. CRUOUT becomes the input to a flip-flop which is decoded to generate CRU select signals. The CRU bit select and $\overline{\text{MEMEN}}$ are gated to the flip-flop's Clear input. Therefore, when CRU bit "n" is selected, $\overline{\text{CLR}}$ is removed and CRUOUT can be clocked through. A set bit (SBO) instruction switches the flip-flop on. As soon as the flip-flop address is removed at the end of the CRU I/O machine cycle, the flip-flop is cleared, thus terminating the interrupt acknowledge pulse.

The logic illustrated in Figure 18-12A requires that you execute an SBO instruction at the beginning of every interrupt service routine in order to generate an interrupt acknowledge. You could require every interrupt service routine to control the length of the interrupt acknowledge pulse by executing an SBZ instruction to terminate the pulse. Figure 18-12B shows logic to implement this scheme. When the flip-flop is selected by the appropriate CRU address, CRUCLK will clock CRUOUT to INT ACK n. At other times, CRUCLK will merely clock the flip-flop's output through, thus making no change. In this way, only SBO and SBZ instructions which address INT ACK n can set or reset the flip-flop.

Figure 18-13 illustrates generation of an interrupt acknowledge signal by identifying specific addresses on the Address Bus. Following any interrupt acknowledge, specific memory locations will be accessed, as identified in Figure 18-11, in order to fetch the new values for the Program Counter and WP register. Figure 18-13 shows a very simple scheme whereby Address Bus lines are combined with $\overline{\text{MEMEN}}$ low to generate high pulses for the duration of a valid address. That is to say, the interrupt acknowledge signal will last for one machine cycle — the time that the valid address exists on the Address Bus.

External logic which requested an interrupt removes its interrupt request and priority signals upon receiving an interrupt acknowledge.

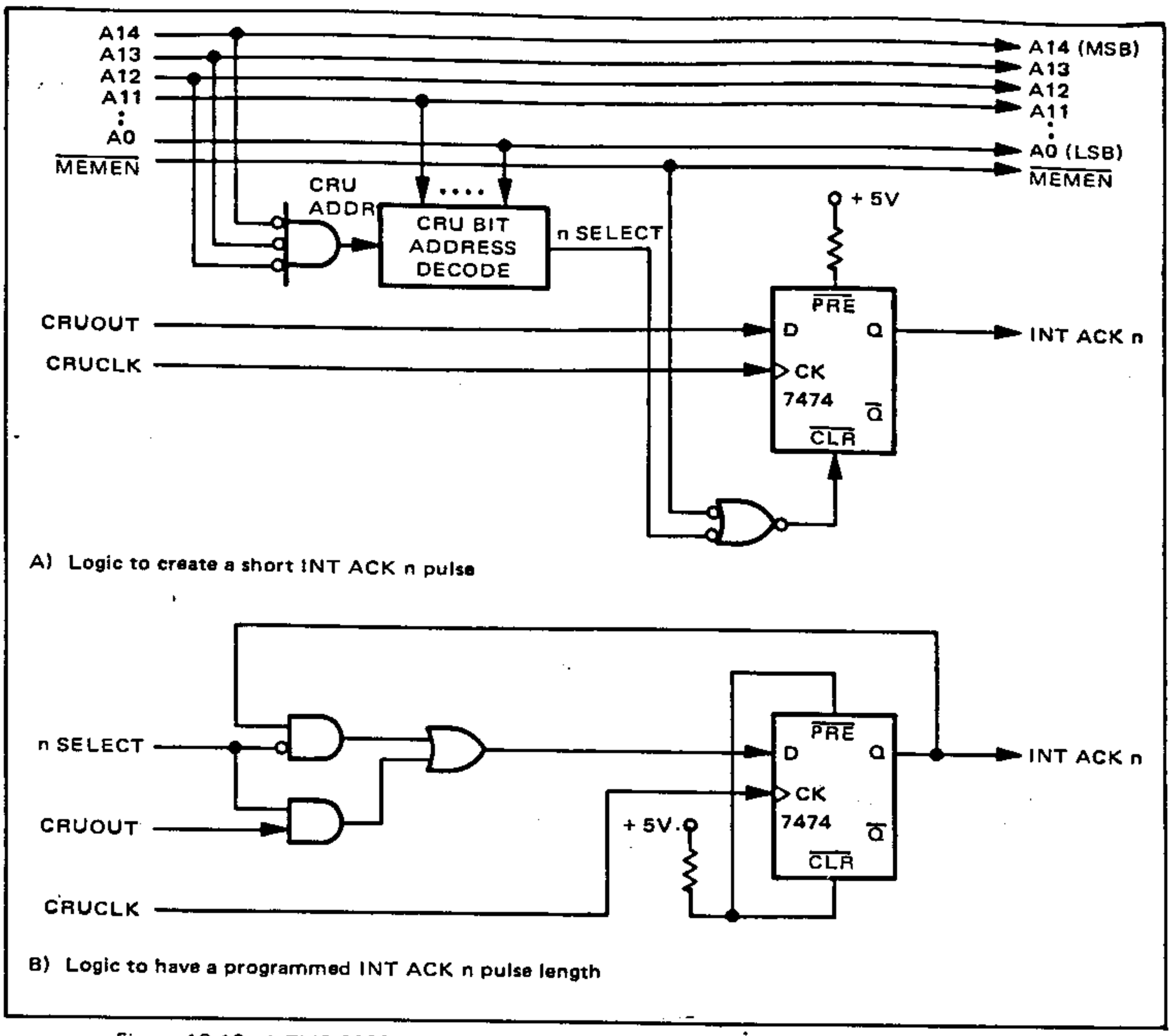


Figure 18-12. A TMS 9900 Interrupt Acknowledge Pulse Generated Using an SBO Instruction

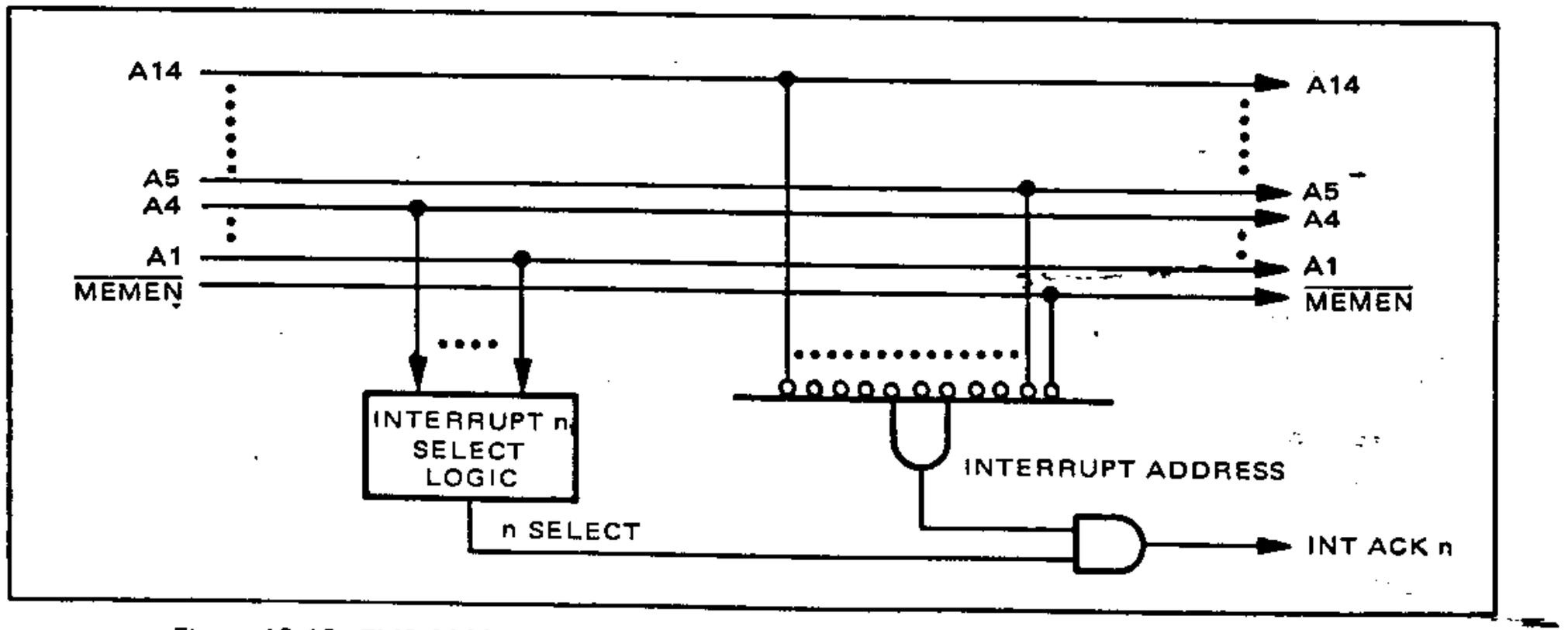


Figure 18-13. TMS 9900 Interrupt Acknowledge Generated by Decoding Valid Addresses

THE TMS 9900 RESET

You reset the 9900 microcomputer system by inputting a low $\overline{\text{RESET}}$ signal. This signal must remain low for at least 3 clock periods. When the low $\overline{\text{RESET}}$ signal is removed, the following machine cycle sequence is executed:

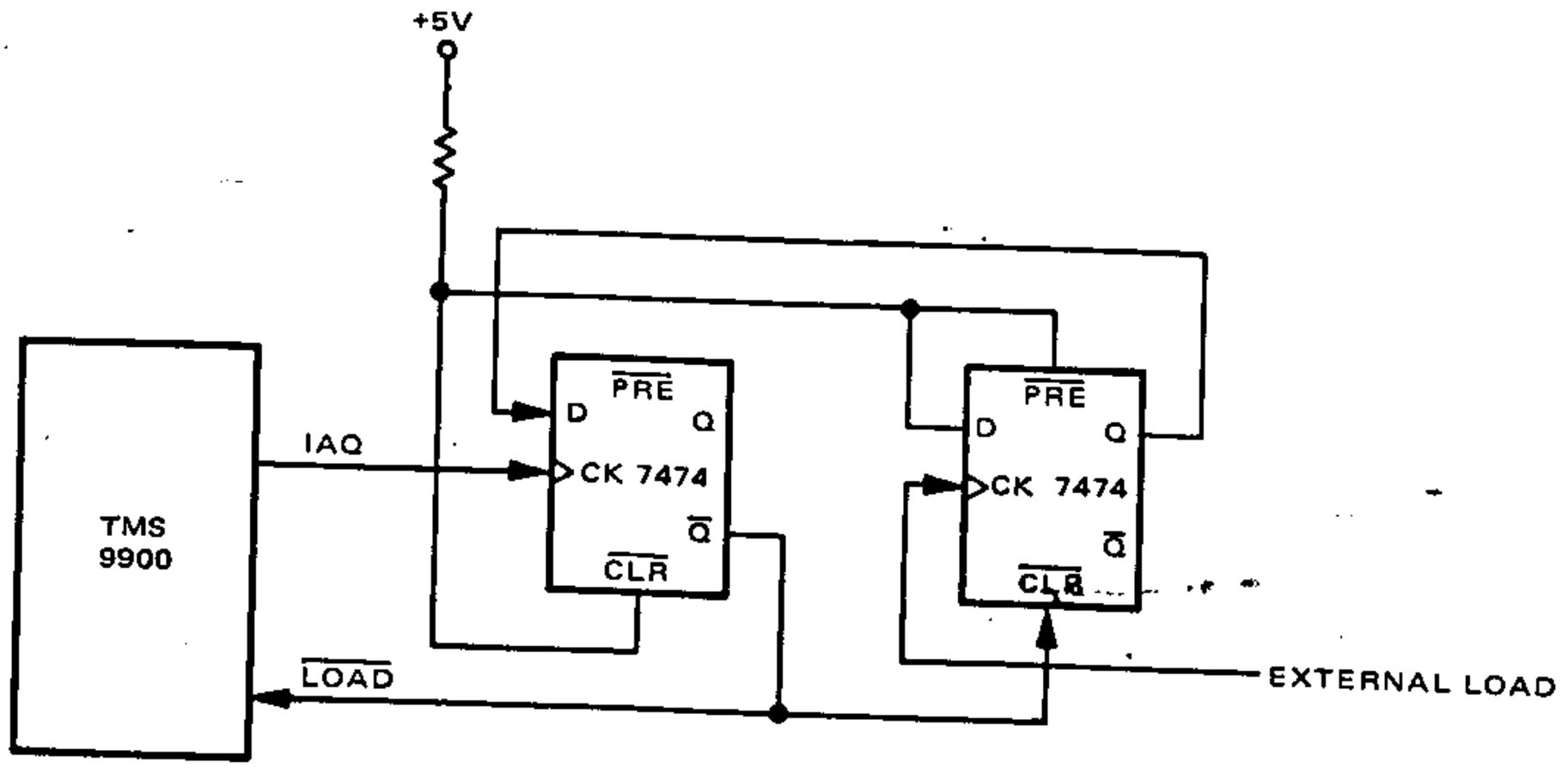
Cycle	Type	Figure	Function
1	ALU	18-3	Prepare for Level 0 interrupt
2	ALU	18-3	
3	ALU	18-3	
4	MEMORY READ	18-4	Fetch new WP register contents from memory word 0000_{16} to temporary storage
5	ALU	18-3	
6	MEMORY WRITE	18-5	Store Status register contents in new R15
7	ALU	18-3	
8	MEMORY WRITE	18-5	Store Program Counter contents in new R14
9	ALU	18-3	
10	MEMORY WRITE	18-5	Store old WP register contents in new R13
11	ALU	18-3	
12	MEMORY READ	18-4	Fetch new Program Counter contents from memory word 0001_{16}
13	ALU	18-3	Load WP register from temporary storage

Thus, program execution begins with a program whose starting address is stored in memory word 1. The starting address for the 16 general purpose registers is stored in memory word 0.

The TMS 9900 has a Reset instruction (RSET). In reality, this instruction resets only the interrupt mask in the Status register; it also outputs a code on the Address Bus, as identified in Table 18-1 and illustrated in Figure 18-8. TM 990 minicomputer systems use this signal to generate a program-initiated Reset. If you are designing your own TMS 9900-based microcomputer system, you are free to use the RSET instruction in any way.

THE TMS 9900 LOAD OPERATION

The $\overline{\text{LOAD}}$ input to the TMS 9900 is a non-maskable, highest priority interrupt. Load must be input low for at least one instruction's duration. Since the length of an instruction can vary, you must use the IAQ signal to control the $\overline{\text{LOAD}}$ input pulse width. Texas Instruments' literature recommends the following circuit:



The CPU checks $\overline{\text{LOAD}}$ at the end of each instruction's execution.

After a valid $\overline{\text{LOAD}}$ input has been acknowledged, the following machine cycle sequence is executed:

Cycle	Type	Figure	Function
1	ALU	18-3	
2	MEMORY READ	18-4	Input new WP register contents from memory word $7FFE_{16}$ to temporary storage
3	ALU	18-3	
4	MEMORY WRITE	18-5	Store in new R15
5	ALU	18-3	
6	MEMORY WRITE	18-5	Store incremented Program Counter contents in new R14
7	ALU	18-3	
8	MEMORY WRITE	18-5	Store old WP register contents in new R13
9	ALU	18-3	
10	MEMORY READ	18-4	Input new Program Counter contents from word $7FFF_{16}$
11	ALU	18-3	Load WP register from temporary storage

There are two differences between Reset and Load. First, the $\overline{\text{RESET}}$ input provides a true hardware reset, synchronizing internal operations, as well as a level 0 interrupt; $\overline{\text{LOAD}}$ provides only a non-maskable interrupt. Second, the Reset vector in bytes 0 through 3, while the Load vector is in bytes $FFFC_{16}$ through $FFFF_{16}$.

In TM 990 minicomputer systems, the LREX instruction is frequently used as a software load. Output due to LREX is identified in Table 18-1 and Figure 18-8. In a TMS 9900 microcomputer system, you can use the LREX signal in any way.

THE TMS 9900 INSTRUCTION SET

The TMS 9900 Instruction set is extremely powerful when compared to any 16-bit microprocessor described in this book. When you consider that the TMS 9900 was first manufactured in 1976, the power of this instruction set becomes more impressive.

With regard to instructions described in Table 18-2, some explanations are required.

The ABS instruction converts the contents of a memory location to their absolute value. That is to say, this instruction assumes that the memory location contains a signed binary number. If the number is positive, nothing happens. If the number is negative, the twos complement of the number is taken.

A number of instructions act on specific bits within source and destination memory words. These include **the SOC, SOCB, SZC, SZCB, COC, and CZC instructions.** In the OPERATION PERFORMED column of Table 18-2, the word "corresponding" means that the source word bits are affected only if selected by the destination word bit pattern. For example, the SOC instruction will be interpreted as follows:

Source:	1	0	1	1	1	0	1	0	0	1	1	0	1	1	0	1
Destination:	1	0	1	1	0	0	1	0	1	0	1	0	0	0	1	0
After SOC:	1	0	1	1	1	0	1	0	1	1	1	0	1	1	1	1

Here are the new destination contents.

This is equivalent to an OR operation.

The SOCB instruction is identical to the SOC instruction, except that only one byte is affected. This may be any memory byte or the high-order byte of a general purpose register.

The SZC instruction may be illustrated as follows:

Source:	1	0	1	0	0	1	1	0	1	0	1	1	1	0	0	1
Destination:	0	1	0	1	1	0	1	1	1	0	1	0	1	1	0	1
After SZC:	0	1	0	1	1	0	0	1	0	0	0	0	0	1	0	0

This is equivalent to complementing the source operand and then ANDing the two operands. The SZCB instruction is identical to the SZC instruction, except that only one byte is affected.

The COC instruction compares Source Register 1 bits with general purpose register bits that happen to be in the same bit positions. If all corresponding general purpose register bits are also 1, then the Equal status is set. Matches are not significant in bit positions if the source register bit is 0.

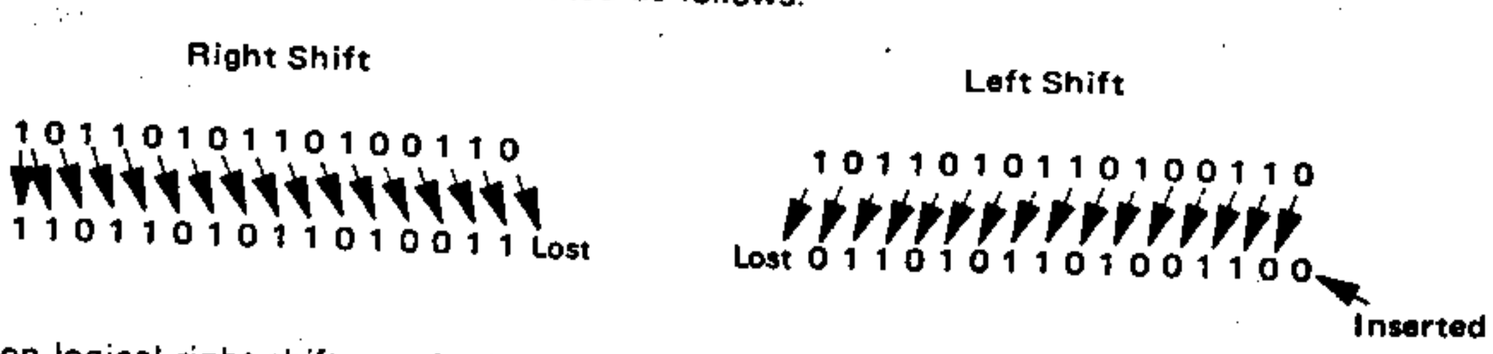
The CZC instruction operates in the same fashion as the COC instruction, except that those source memory word bits that are 0 become significant. That is to say, if every source memory word 0 bit has a corresponding Workspace register 0 bit, then the Equal status is set. Matches are not significant in bit positions if the source register bit is 1.

The BLWP instruction is a subroutine call accompanied by a context switch. The operand memory address identifies the first of two memory words within which the new WP register and Program Counter contents will be stored.

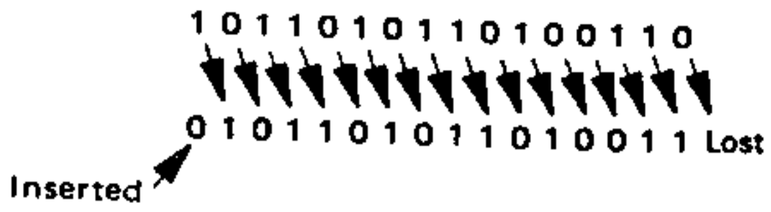
The BLWP instruction is remarkably powerful. The subroutine call and passing parameters to the subroutine become a single operation. The memory words that are to serve as subroutine general purpose registers can be used as general data memory locations prior to the subroutine call. Thus, the subroutine finds its registers pre-loaded with data when it starts executing.

The RTWP instruction should be used to return from a subroutine that is called by the BLWP instruction.

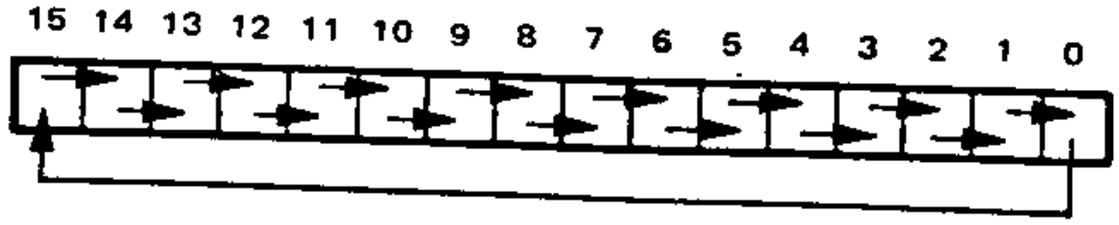
One-bit position arithmetic shifts may be illustrated as follows:



A one-bit-position logical right shift may be illustrated as follows:



A one-bit right rotate (Shift Right Circular) may be illustrated as follows:



You can specify any number of bits, from 1 to 15, as the number of bit positions for any TMS 9900 shift or rotate instruction. If you specify 0 for the bit count, then the actual bit count is taken from the four low-order bits of general purpose Register R0. If these four low-order bits are 0000, then the bit count is assumed to be 16.

The following symbols are used in Table 18-2:

AG	Arithmetic Greater Than status
C	Carry status
CNT	4-bit count field
CRUA	CRU base address from R12
d	Destination memory word. There are five possible options for the destination memory word. They are represented by these combinations of addressing modes: Workspace Register D Implied through Workspace Register D Direct address Direct, indexed address Implied through Workspace Register D, auto-increment Workspace Register D
DATA4	4-bit data unit
DATA16	16-bit data unit
DISP	8-bit signed displacement
EQ	Equal status bit of Status register
G	Both the AG and LG statuses
LG	Logical Greater Than status
OP	Odd Parity status
OV	Overflow status
PC	Program Counter
R	Any of the 16 Workspace registers
Rxx	Workspace register. For example, R15 is Workspace Register 15
S	Source memory location. Addressing options identical to destination memory location
ST	Status register
WP	Workspace Pointer register
x<y,z>	Bits y through z of the quantity x. For example, ([S] * [R])<31,16> represents the high-order word of the product of the contents of the Source Register S and the Workspace Register R.
[ ]	Contents of location enclosed within brackets. If a register designation is enclosed within the brackets, then the designated register's contents are specified. If a memory address is enclosed within the brackets, then the contents of the addressed memory location are specified.
*	Multiplication
/	Division
Λ	Logical AND
V	Logical OR
⊻	Logical Exclusive-OR
←	Data is transferred in the direction of the arrow

Under the heading of STATUSES in Table 18-2, an X indicates statuses which are modified in the course of the instruction's execution. If there is no X, it means that the status maintains the value it had before the instruction was executed.

Byte-operand instructions will affect half of a 16-bit memory word. If the word is accessed as a general purpose register, then only the high-order byte will be affected. If the word is accessed as non-register memory, then the byte affected is determined by the least significant bit of the 16-bit address: 0 selects the high-order byte; 1 selects the low-order byte.

Table 18-2. TMS 9900 Instruction Set Summary

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				G	EQ	C	OV	OP		
I/O	LDCR	S,CNT	2	X	X			X*		[CRUA]←[S<CNT-1,0>] Transfer the specified number of bits from source memory word to the CRU.
	STCR	D,CNT	2	X	X			X*		[D<CNT-1,0>]←[CRUA] Transfer the specified number of bits from the CRU to destination memory word.
	SBO	DISP	2							[CRUA + DISP]← 1 Set bit in CRU to 1.
	SBZ	DISP	2							[CRUA + DISP]← 0 Set bit in CRU to 0.
	TB	DISP	2		X					If [CRUA + DISP] = 0, then [EQ] = 1; or else [EQ] = 0 Test bit in CRU.
PRIMARY MEMORY REFERENCE	MOV	S,D	2	X	X					[D]←[S] 16-bit move contents of source memory word to destination memory word.
	MOVB	S,D	2	X	X			X		[D]←[S] 8-bit move contents of source memory byte to destination memory byte.
SECONDARY MEMORY REFERENCE (MEMORY OPERATE)	A	S,D	2	X	X	X	X			[D]←[S] + [D] 16-bit add contents of source memory word to contents of destination memory word.
	AB	S,D	2	X	X	X	X	X		[D]←[S] + [D] 8-bit add contents of source memory byte to contents of destination memory byte.
	S	S,D	2	X	X	X	X			[D]←[D] - [S] 16-bit subtract contents of source memory from contents of destination memory word.
	SB	S,D	2	X	X	X	X	X		[D]←[D] - [S] 8-bit subtract contents of source memory byte from contents of destination memory byte.
	C	S,D	2	X	X					Set status flags based on 16-bit comparison of source and destination memory word contents.
	CB	S,D	2	X	X			X		Set status flags based on 8-bit comparison of source memory byte contents and destination memory byte contents.
	XOR	S,R	2	X	X					[R]←[S] ∨ [R] Exclusive-OR contents of source memory word with Workspace Register R.
	MPY	S,R	2							[R]←[([S]*[R])<31,16>] [R + 1]←[([S]*[R])<15,0>] Multiply the contents of source memory word by contents of Workspace Register R. Store most significant word of result in R. Store least significant word of result in Workspace Register R + 1.
	DIV	S,R	2				X			[R]←([R,R + 1]/[S])(quotient) [R + 1]←([R,R + 1]/[S])(remainder) Divide the 32-bit quantity represented by R (high-order word) concatenated with R + 1 (low order) by the contents of the source memory word. Store the quotient in R, the remainder in R + 1 and set overflow if quotient will exceed 16 bits.
	INC	D	2	X	X	X	X			[D]←[D] + 1 Increment contents of memory word by 1.
	INCT	D	2	X	X	X	X			[D]←[D] + 2 Increment contents of memory word by 2.
	DEC	D	2	X	X	X	X			[D]←[D] - 1 Decrement contents of memory word by 1.

*OP status is affected only if between 1 and 8 bits are transferred.

Table 18-2. TMS 9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				G	EQ	C	OV	OP		
SECONDARY MEMORY REFERENCE (MEMORY OPERATE) (CONTINUED)	DECT	D	2	X	X	X	X			[D]—[D] - 2 Decrement contents of memory word by 2.
	CLR	D	2							[D]—0000_{16} Clear the destination memory word.
	SETO	D	2							[D]—$FFFF_{16}$ Set all bits of memory word.
	INV	D	2	X	X					[D]—$\overline{[D]}$ Ones complement the destination memory word.
	NEG	D	2	X	X	X	X			[D]—$\overline{[D]} + 1$ Twos complement the destination memory word.
	ABS	D	2	X	X	X	X			[D]—| [D] | Take the absolute (unsigned) value of the destination memory word's contents.
	SWPB	D	2							[D<15,8>]⟷[D<7,0>] Exchange the high and low bytes of the memory word.
	SOC	S,D	2	X	X					If [S<i>]=1, then [D<i>]—1 Set the bits in the destination memory word that correspond to 1s in the source memory word for all 16 bits.
	SOCB	S,D	2	X	X			X		If [S<i>]=1, then [D<i>]—1 Set the bits in the destination memory word that correspond to 1s in the source memory word for 8 bits.
	SZC	S,D	2	X	X					If [S<i>]=1, then [D<i>]—0 Clear the bits in the destination memory word that correspond to 1s in the source memory word for all 16 bits.
	SZCB	S,D	2	X	X			X		If [S<i>]=1, then [D<i>]—0 Clear the bits in the destination memory word that correspond to 1s in the source memory word for 8 bits.
	COC	S,R	2		X					If for all [S<i>]=1, [R<i>]=1, then [EQ]—1 If the bits in the Workspace Register R that correspond to the set bits in the source memory word are all 1s, set the EQUAL status.
	CZC	S,R	2		X					If for all [S<i>]=1, [R<i>]=0, then [EQ]=1 If the bits in the Workspace Register R that correspond to set bits in the source memory word are all 0s, set the EQUAL status.
IMMEDIATE	LI	R,DATA16	4	X	X					[R]—DATA16 Load immediate to Workspace Register R.
	LWPI	DATA16	4							[WR]—DATA16 Load immediate to Workspace Pointer Register, WR.

Table 18-2 TMS 9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				G	EQ	C	OV	OP		
IMMEDIATE OPERATE	CI	R,DATA16	4	X	X					Set the status flags based on 16-bit comparison between contents of Workspace Register R and immediate data. [R]←[R] + DATA16 Add immediate to Workspace Register R contents. [R]←[R] Λ DATA16 AND immediate with Workspace Register R contents. [R]←[R] V DATA16 OR immediate with Workspace Register R contents.
	AI	R,DATA16	4	X	X	X	X			
	ANDI	R,DATA16	4	X	X					
	ORI	R,DATA16	4	X	X					
JUMP	B	S	2							[PC]←[S] Branch unconditional to address in Source memory word. [PC]←[PC] + DISP Branch unconditional.
	JMP	DISP	2							
SUBROUTINE CALL AND RETURN	BL	S	2							[R11]←[PC] + 1 [PC]←[S] Branch to subroutine at address in source memory word. [R13]←[WP] [R14]←[PC] [R15]←[ST] [WP]←[S] [PC]←[S + 2] Branch to subroutine whose address is stored in source memory word + 1. Perform context switch to R0 address contained in source memory word. [WP]←[R13] [PC]←[R14] [ST]←[R15] Perform a backward context switch.
	BLWP	S	2							
	RTWP		2	X	X	X	X	X		
BRANCH ON CONDITION	JEQ	DISP	2							If [EQ]=1; then [PC]←[PC] + DISP Branch if equal. If [EQ]=0; then [PC]←[PC] + DISP Branch if not equal. If [AG]=1; then [PC]←[PC] + DISP Branch on arithmetic greater than. If [AG]=0 and [EQ]=0; then [PC]←[PC] + DISP Branch on arithmetic less than. If [LG]=1 or [EQ]=1; then [PC]←[PC] + DISP Branch on logical greater than or equal. If [LG]=1 and [EQ]=0; then [PC]←[PC] + DISP Branch on logical greater than. If [LG]=0 and [EQ]=0; then [PC]←[PC] + DISP Branch on logical less than. If [EQ]=1 or [LG]=0; then [PC]←[PC] + DISP Branch on less than or equal.
	JNE	DISP	2							
	JGT	DISP	2							
	JLT	DISP	2							
	JHE	DISP	2							
	JH	DISP	2							
	JL	DISP	2							
	JLE	DISP	2							

Table 18-2. TMS 9900 Instruction Set Summary (Continued)

TYPE	MNEMONIC	OPERAND(S)	BYTES	STATUSES						OPERATION PERFORMED
				G	EQ	C	OV	OP		
BRANCH ON CONDITION (CONTINUED)	JNC	DISP	2							If [C]=0; then [PC]←[PC] + DISP Branch on carry reset.
	JNO	DISP	2							If [OV]=0; then [PC]←[PC] + DISP Branch on overflow reset.
	JOC	DISP	2							If [C]=1; then [PC]←[PC] + DISP Branch on carry set.
	JOP	DISP	2							If [OP]=1; then [PC]←[PC] + DISP Branch on odd parity set.
REGISTER OPERATE	SLA	R,CNT	2	X	X	X	X			Arithmetic shift the Workspace Register R left the specified number of bits.
	SRA	R,CNT	2	X	X	X				Arithmetic shift the Workspace Register R right the specified number of bits.
	SRL	R,CNT	2	X	X	X				Logical shift the Workspace Register R right the specified number of bits.
	SRC	R,CNT	2	X	X	X				Rotate the Workspace Register R right the specified number of bits.
STATUS AND INTERRUPT	STST	R	2							[R]←[ST] Store the Status register into Workspace Register R.
	STWP	R	2							[R]←[WP] Store the Workspace Pointer into Workspace Register R.
	LIMI	DATA4	4							[SR<3,0>] ←DATA4 Load immediate data into the interrupt mask bits of the Status register.
	XOP	S,R	2					X		[R13]←[WP] [R14]←[PC] [R15]←[ST] [R11]←[S] [WP]←[40_{16} + (4*[R])] [PC]←[41_{16} + (4*[R])] Perform a context switch. This is the software interrupt.
EXECUTE	X	S	2							Execute the instruction represented by the data in the source location. If that instruction has immediate operand words, those words must be located directly after the X instruction. The instruction [S] will affect the status flags but its fetch will not cause IAQ to go high.
EXTERNALLY DEFINED	IDLE		2							CPU enters Halt state.
	RSET									CPU clears interrupt mask and outputs 001 on three high-order Address Bus lines.
	CKOF									011 on three high-order Address Bus lines.
	CKON									110 out on three high-order Address Bus lines.
	LREX									101 out on three high-order Address Bus lines. 111 out on three high-order Address Bus lines.

THE BENCHMARK PROGRAM

For the TMS 9900, our benchmark program may be illustrated as follows:

```
        BLWP    MOVE                    CONTEXT SWITCH TO APPROPRIATE REGISTERS
        -
        -
        -
LOOP    MOV     @IOBUF(R1),*R2+         LOAD NEXT INPUT WORD IN NEXT TABLE WORD
        DEC     R1                      DECREMENT COUNT
        JNE     LOOP                    RETURN FOR MORE
        RTWP                            RETURN FROM SUBROUTINE
```

Let us look at how our benchmark program can collapse to just five instructions.

We assume that there is some set of 16 General Purpose registers within which we store the word count and the address of the first free word in TABLE. We illustrated this idea when describing context switching earlier in the chapter.

Observe that Register R1 contains the word count, and is therefore used as an Index register, while Register R2 addresses the first free word in TABLE. Note that the contents of Register R2 are incremented automatically when the next byte is loaded into the table.

The BLWP instruction will branch to the program which performs the required data move, but simultaneously it loads the Workspace register with the appropriate initial address. We do not need to load any initial addresses or word counts into registers, since we have adopted the memory space where this data is stored to serve as our General Purpose registers.

After the move has been completed, we do not have to update any counters or pointers, because they were updated "in situ". All we have to do upon completing the move is store the contents of the current General Purpose Registers 13 and 14 to the Workspace register and Program Counter.

The following notation is used in Table 18-3:

aa	Two bits determining the addressing mode for the destination memory word
bb	Two bits determining the addressing mode for the source memory word
cccccccc	8-bit signed address displacement
dddd	Four bits used with aa to determine the destination memory word
eeee	4-bit count field
rrrr	Four bits choosing the Workspace register
ssss	Four bits used with bb to determine the source memory word
xx	16 bits of immediate data

If either aa or bb is 10_2, and the corresponding register specified is 0_2, then an additional 16-bit direct memory address word, used in computing the effective memory address of the operand, will follow the instruction.

If aa and bb are 10_2, and both corresponding register specifications are 0, then two additional 16-bit direct memory addressing words will follow the instruction: the first will be used in computing the source address, the second will be used in computing the destination address.

Table 18-3. TMS 9900 Instruction Set Object Codes

INSTRUCTION		OBJECT CODE	BYTES	CLOCK PERIODS*
A	S,D	1010aaddddbbssss	2	14-30 (1)
AB	S,D	1011aaddddbbssss	2	14-30 (1)
ABS	D	0000011101aadddd	2	12-20 (6)
AI	R,DATA16	0000000100010rrrr XX	4	14 (17)
ANDI	R,DATA16	0000000100100rrrr XX	4	14 (17)
B	S	0000010001bbssss	2	8-16 (7)
BL	S	0000011010bbssss	2	12-20 (9)
BLWP	S	0000010000bbssss	2	26-34 (10)
C	S,D	1000aaddddbbssss	2	14-30 (1)
CB	S,D	1001aaddddbbssss	2	14-30 (1)
CI	S,D	0000000101000rrrr XX	4	14 (18)
CKON		0000000111000000	2	6 (14)
CKOF		0000000110100000	2	6 (14)
CLR	D	0000010011aadddd	2	10-18 (5)
COC	S,R	001000rrrrbbssss	2	10-18 (1)
CZC	S,R	001001rrrrbbssss	2	14-22 (1)
DEC	D	0000011000aadddd	2	14-22 (5)
DECT	D	0000011001aadddd	2	10-18 (5)
DIV	S,R	001111rrrrbbssss	2	10-18 (3)
IDLE		0000000101000000	2	6 (14)
INC	D	0000010110aadddd		16-124 (5)
INCT	D	0000010111aadddd	2	10-18 (5)
INV	D	0000010101aadddd	2	10-18 (5)
JEQ	DISP	00010011cccccccc	2	10-18 (15)
JGT	DISP	00010101cccccccc	2	8/10 (15)
JH	DISP	00011011cccccccc	2	8/10 (15)
JHE	DISP	00010100cccccccc	2	8/10 (15)
JL	DISP	00011010cccccccc	2	8/10 (15)
JLE	DISP	00010010cccccccc	2	8/10 (15)
JLT	DISP	00010001cccccccc	2	8/10 (15)
JMP	DISP	00010000cccccccc	2	10 (15)
JNC	DISP	00010111cccccccc	2	8/10 (15)
JNE	DISP	00010110cccccccc	2	8/10 (15)
JNO	DISP	00011001cccccccc	2	8/10 (15)
JOC	DISP	00011000cccccccc	2	8/10 (15)
JOP	DISP	00011100cccccccc	2	8/10(15)
LDCR	S,CNT	001100ccccbbssss	2	22-52 (11)
LI	R,DATA16	0000001000000rrrr XX	4	12 (19)
LIMI	DATA4	0000001100000000 XX	4	16 (21)
LREX		0000001111100000	2	6 (14)
LWPI	DATA16	0000001011100000 XX	4	10 (20)
MOV	S,D	1100aaddddbbssss	2	14-30 (1)
MOVB	S,D	1101aaddddbbssss	2	14-30 (1)
MPY	S,R	001110rrrrbbssss	2	52-60 (2)
NEG	D	0000010100aadddd	2	12-20 (5)
ORI	R,DATA16	0000001001110rrrr XX		14 (17)
RSET		0000001101100000	2	6 (14)
RTWP		0000001110000000	2	14 (8)
S	S,D	0110aaddddbbssss	2	14-30 (1)
SB	S,D	0111aaddddbbssss	2	14-30 (1)
SBO	DISP	00011101cccccccc	2	12 (13)
SBZ	DISP	00011110cccccccc	2	12 (13)
SETO	D	0000011100aadddd	2	10-18 (5)
SLA	R,CNT	00001010ccccrrrr	2	14-52 (16)
SOC	S,D	1110aaddddbbssss	2	14-30 (1)
SOCB	S,D	1111aaddddbbssss	2	14-30 (1)
SRA	R,CNT	00001000ccccrrrr	2	14-52 (16)
SRC	R,CNT	00001011ccccrrrr	2	14-52 (16)
SRL	R,CNT	00001001ccccrrrr	2	14-52 (16)
STCR	D,CNT	001101ccccaadddd	2	42-60 (12)
STST	R	0000001011001rrr	2	8 (23)
STWP	R	0000001010101rrr	2	8 (22)
SWPB	D	0000011011aadddd	2	10-18 (23)
SZC	S,D	0100aaddddbbssss	2	14-30 (1)
SZCB	S,D	0101aaddddbbssss	2	14-30 (1)
TB	DISP	00011111cccccccc	2	12 (8)
X	S	0000010010bbssss	2	8-16 (7)
XOP	S,R	001011rrrrbbssss	2	44-52 (4)
XOR	S,R	001010rrrrbbssss	2	14-22 (1)

* The number in brackets identifies the instruction's machine cycle sequence, as defined in the preceding text.

The minimum and maximum number of clock periods for the execution of each instruction are shown in the CLOCK PERIODS column of Table 18-3. Remember that a machine cycle consists of two clock periods. The bracketed number after the number of clock periods identifies the machine cycle sequence. Machine cycle sequences associated with each bracketed number are listed below. In the machine cycle list below, the following abbreviations are used:

R represents a memory read machine cycle as identified in Figure 18-4.

A represents an ALU machine cycle as illustrated in Figure 18-3.

W represents a memory write machine cycle as illustrated in Figure 18-5.

C represents a CRU machine cycle as illustrated in Figures 18-6 and 18-7.

A subscript associated with any machine cycle notation identifies that machine cycle repeated a number of times. Thus A_3 is equivalent to -A-A-A-.

M represents memory address computation machine cycles. Memory address computations were described earlier in this chapter. In summary, here are the various possibilities for M:

Register addressing:	R
Implied memory addressing:	R-A-R
Implied memory addressing with auto-increment (for byte operand):	R-A-W-R
Implied memory addressing with auto-increment (for word operand):	R-A-A-W-R
Direct addressing:	A-A-R-A-R
Direct, indexed addressing:	R-A-R-A-R

(1) R-A-M-A-M-A-W
(2) R-A-M-A-R-A_{18}-W-A-W
(3) R-A-M-A-R-A-A-R-A_x-W-A-W ($51 \leqslant x \leqslant 35$)
(4) R-A-M-A_3-R-A-W-A-W-A-W-A-W-A-R-A
(5) R-A-M-A-W
(6) R-A-M-A_3-W-A
(7) R-A-M-A
(8) R-A-A-R-R-R-A
(9) R-A-M-A-A-W
(10) R-A-M-A-A-W-A-W-A-W-A-R-A
(11) R-A-M-A_4-R-A-C_x-A ($16 \leqslant x \leqslant 1$)
(12) R-A-M-A-R-A-A-C_x-A_y-W ($16 \leqslant x \leqslant 1$, $11 \leqslant y \leqslant 5$)
(13) R-A-A-R-A-C
(14) R-A-A-C-A-A
(15) R-A_x (x=3 or 4)
(16) R-A-R-A-A-R-A_x-W-A ($18 \leqslant x \leqslant 3$)
(17) R-A-A-R-R-A-W
(18) R-A-R-A-R-A-A
(19) R-A-A-R-A-W
(20) R-A-A-R-A
(21) R-A-A-R-A_3
(22) R-A-A-W
(23) R-A-M-A-R-A_4-W

THE TMS 9980A AND THE TMS 9981 MICROPROCESSORS

The TMS 9980A and the TMS 9981 are low-cost variations of the TMS 9900. The principal differences between the TMS 9900 series and TMS 9980 series microprocessors are summarized in Table 18-4. Differences between the TMS 9980A and the TMS 9981 are summarized in Table 18-5.

This discussion of the TMS 9980 series microprocessors covers only differences as compared to the TMS 9900.

The TMS 9980 series microprocessors are manufactured using N-channel silicon gate MOS technology. They are packaged as 40-pin DIPs. The TMS 9980A uses three power supplies: -5V, +5V, and +12V. The TMS 9981 uses two power supplies: +5V and +12V.

Typically, a clock cycle time of 400 nanoseconds will be used with TMS 9980 series microprocessors. This generates instruction execution times ranging between 4 and 14 microseconds.

Figure 18-14 illustrates that part of general microcomputer system logic which is implemented by the TMS 9980 series microprocessors. This figure is identical to Figure 18-1. with the exception of clock logic. which is now shown present.

Programmable registers are implemented and used in exactly the same way the TMS 9900 and TMS 9980 series microprocessors. Note. however. that the **TMS 9980 series microprocessors address a 2048-bit CRU;** therefore. bits 1 through 11 of Register R12 identify the origin of any CRU bit field. The TMS 9900 uses bits 1 through 12 of Register R12 to identify the CRU origin within a 4096-bit CRU.

Table 18-4. A Summary of Differences Between the TMS 9900 and TMS 9980 Series Microprocessors

FUNCTION	TMS 9900	TMS 9980A/TMS 9981
Addressable external memory	32,768 x 16-bit words	16,384 x 8-bit words
DIP pins	64	40
Data Bus	16 bits	8 bits
Address Bus	15 bits	13 bits
External interrupt priorities	15	4
CRU field width	4096 bits	2048 bits
Clock logic	Four external inputs	One external input or internal (TMS 9981 only)

Table 18-5. A Summary of Differences Between the TMS 9980A and TMS 9981 Microprocessors

FUNCTION	TMS 9980A	TMS 9981
Power supplies	-5V. +5V. +12V	+5V. +12V
Clock logic	One external input	One external input or crystal only
Pin incompatibility ties	D0 - D7. INT0 - INT2. $\overline{\Phi 3}$	

The TMS 9980 series microprocessors have a 14-line Address Bus, used to address up to 16,384 bytes of memory. In contrast. the TMS 9900 addresses up to 32,768 16-bit words of external memory. Thus. TMS 9980 programs address memory as bytes. while externally generated addresses also select bytes. The TMS 9900. by way of contrast. addresses memory as bytes within the CPU. but as 16-bit words externally.

The TMS 9980 series microprocessors use exactly the same memory and CRU addressing techniques as the TMS 9900. General-purpose registers are used in the same way. and instruction object codes are identical.

The Status register and Status flags used by the TMS 9980 series microprocessors are identical to those which we have already described for the TMS 9900.

TMS 9980 SERIES MICROPROCESSOR PINS AND SIGNALS

Figure 18-15 illustrates pins and signals for the TMS 9980A. Figure 18-16 provides the same information for the TMS 9981. In both of these illustrations. signal names conform to Texas Instruments nomenclature. For the Data and Address Busses. our notation is given in brackets. Differences result from the fact that we number bits from right to left (0 being the low-order bit). while Texas Instruments numbers bits from left to right (0 becomes the high-order bit). **TMS 9980A/TMS 9981 pin-out differences are shaded in Figures 18-15 and 18-16 so that you can identify them quickly.**

For descriptions of the individual signals, refer to the earlier TMS 9900 discussion.

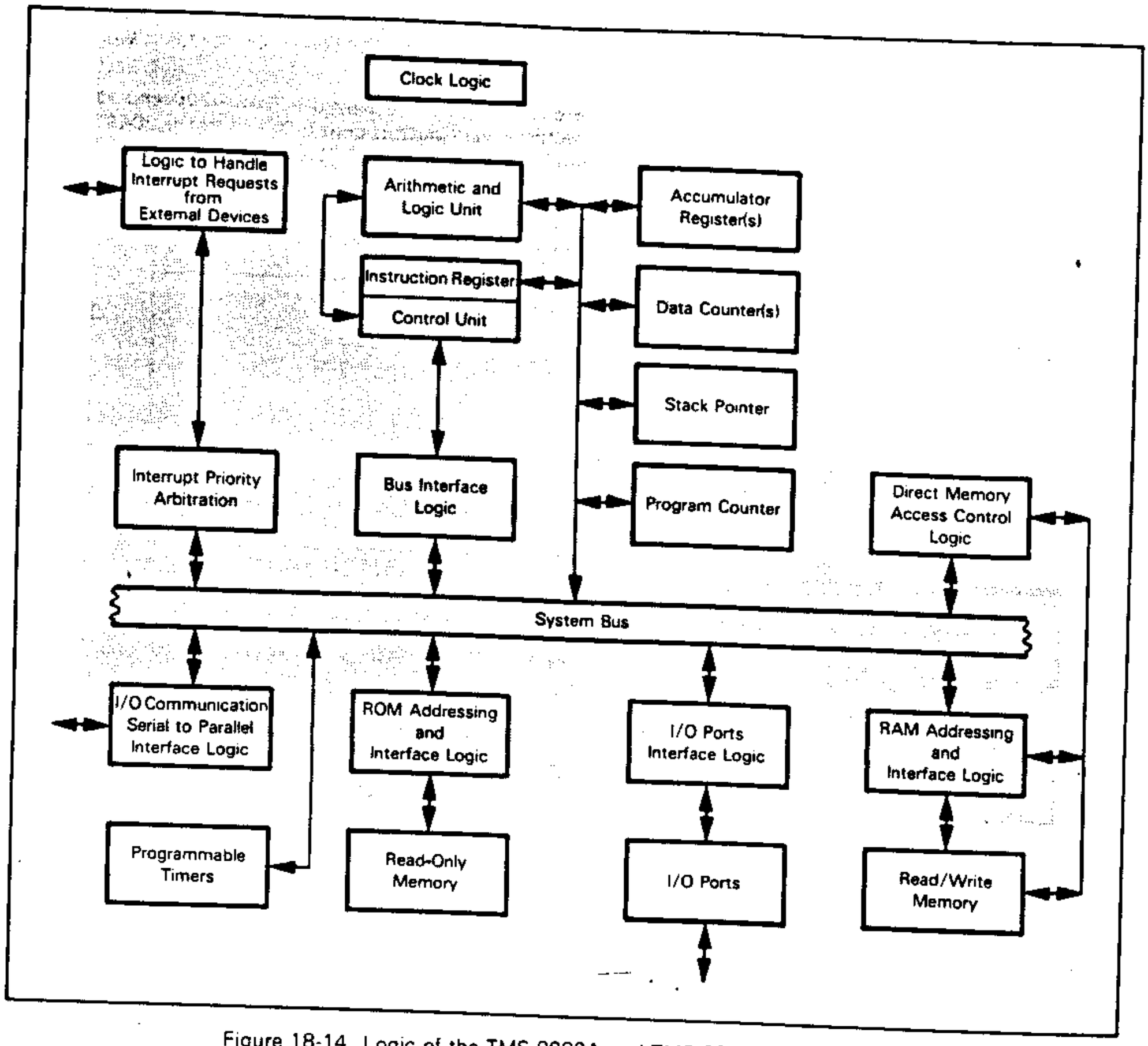
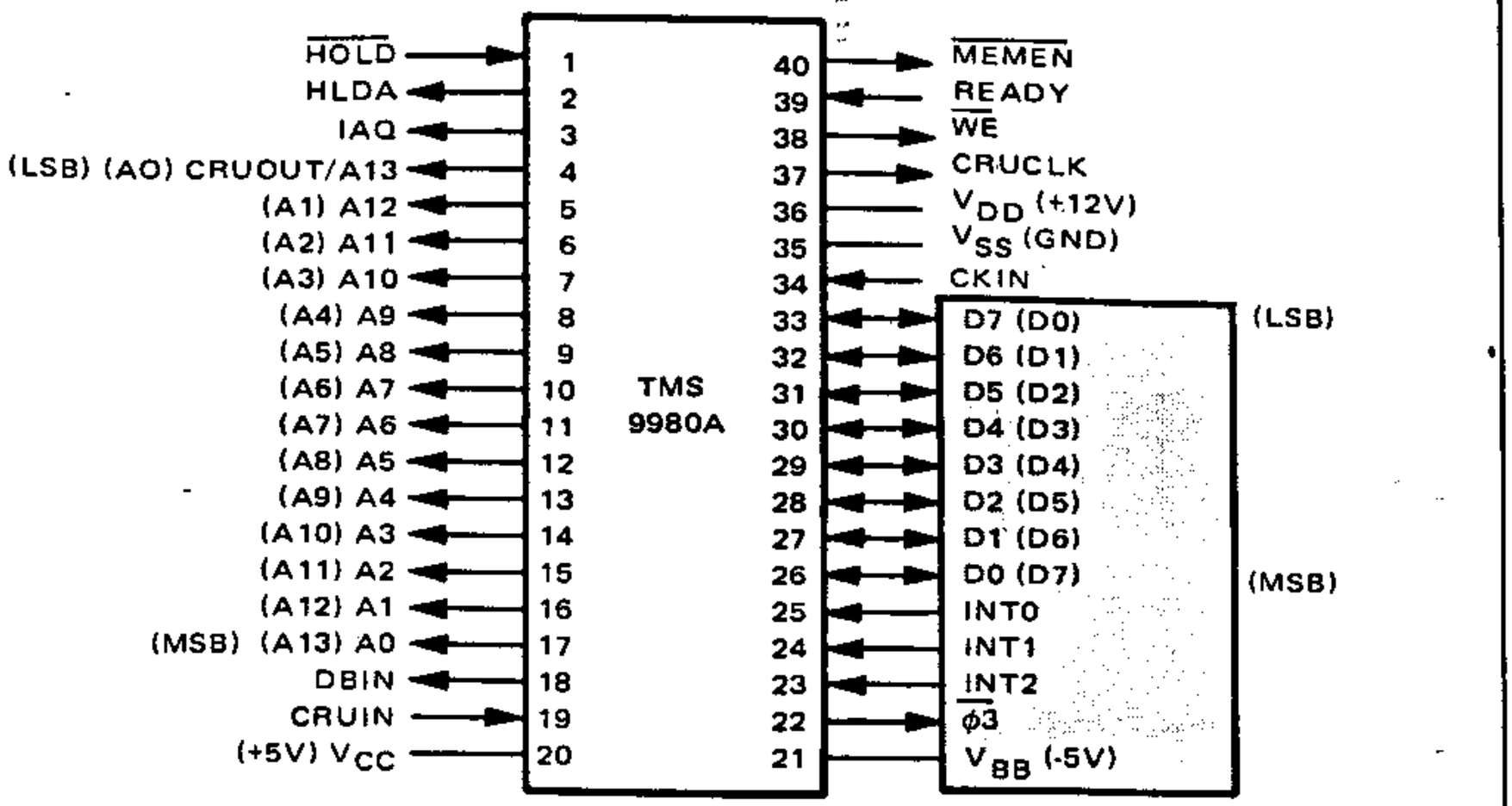
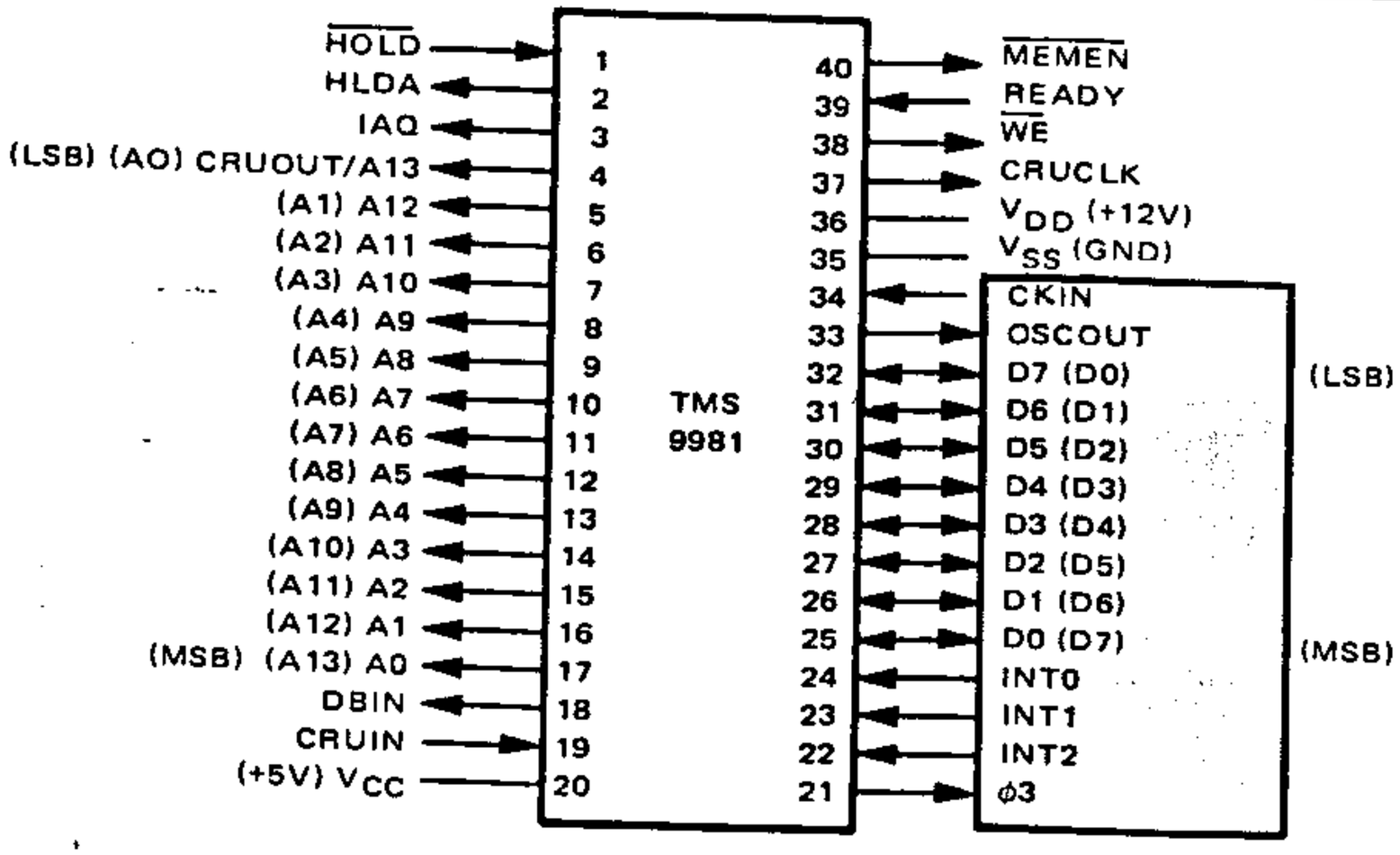


Figure 18-14. Logic of the TMS 9980A and TMS 9981 Microprocessors



Pin Name	Description	Type
A0-A13	Address Bus	Tristate, output
D0-D7	Data Bus	Tristate, bidirectional
CKIN	Clock signal in	Input
$\overline{\phi 3}$	Synchronizing clock	Output
$\overline{\text{MEMEN}}$	Memory Enable	Tristate, output
IAQ	Instruction Fetch	Output
DBIN	Data Bus in	Tristate, output
$\overline{\text{WE}}$	Write Enable	Tristate, output
READY	Memory Ready	Input
WAIT	Wait State indicator	Output
CRUCLK	I/O clock	Output
CRUOUT	Serial I/O out	Output
CRUIN	Serial I/O in	Input
INT0, INT1, INT2	Interrupt request and priority	Input
$\overline{\text{HOLD}}$	DMA request	Input
HOLDA	Hold acknowledge	Output
V_{BB}, V_{CC}, V_{DD}, V_{SS}	Power and Ground reference	

Figure 18-15. TMS 9980A Signals and Pin Assignments



Pin Name	Description	Type
A0-A13	Address Bus	Tristate, output
D0-D7	Data Bus	Tristate, bidirectional
CKIN	Clock or crystal connection	Input
OSCOUT	Crystal connection	Output
$\overline{\phi 3}$	Synchronizing clock	Output
$\overline{\text{MEMEN}}$	Memory Enable	Tristate, output
IAQ	Instruction Fetch	Output
DBIN	Data Bus in	Tristate, output
$\overline{\text{WE}}$	Write Enable	Tristate, output
READY	Memory Ready	Input
WAIT	Wait State indicator	Output
CRUCLK	I/O clock	Output
CRUOUT	Serial I/O out	Output
CRUIN	Serial I/O in	Input
INT0, INT1, INT2	Interrupt request and priority	Input
$\overline{\text{HOLD}}$	DMA request	Input
HOLDA	Hold acknowledge	Output
V_{CC}, V_{DD}, V_{SS}	Power and Ground reference	

Figure 18-16 TMS 9981 Signals and Pin Assignments

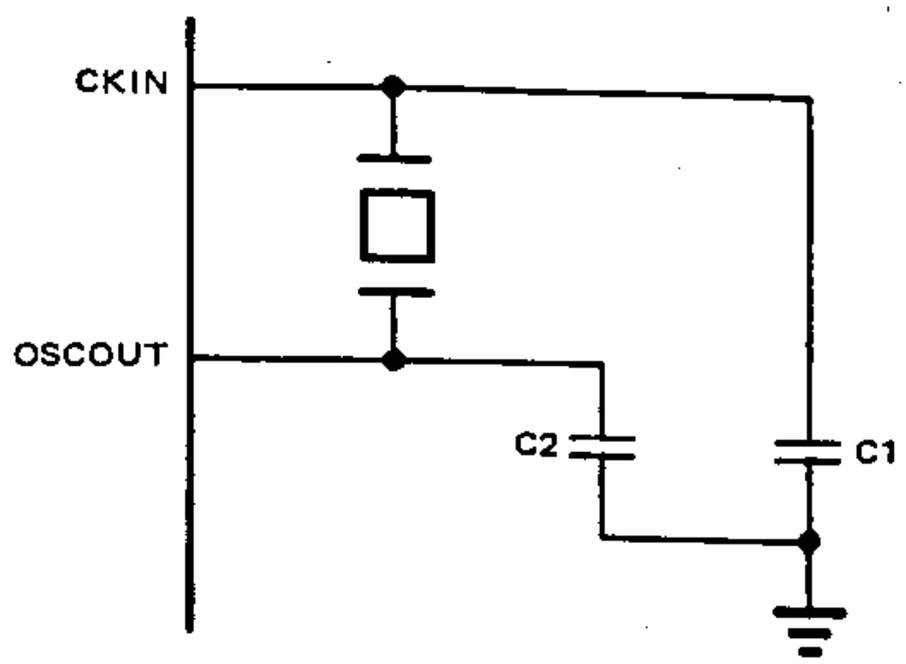
TMS 9980 SERIES MICROPROCESSOR TIMING AND INSTRUCTION EXECUTION

The TMS 9980A and TMS 9981 microprocessors have the same signal relationships and instruction execution sequences as the TMS 9900. The few minor waveform differences are identified in the data sheets at the end of this chapter.

The only significant difference between the TMS 9900 and TMS 9980 series is in clock logic. The TMS 9900 requires four clock inputs, as identified in Figure 18-3.

TMS 9980 SERIES CLOCK LOGIC

The TMS 9980A requires a single clock signal, input via CKIN. The frequency of this clock input must be four times the desired clock frequency. That is to say, CKIN will be divided by four in order to create one clock period. The TMS 9981 can operate with the same CKIN input as the TMS 9980A; however, you can also connect a crystal across CKIN and OSCOUT. This may be illustrated as follows:



C1 and C2 must have values between 10pf and 25pf, typically 15pf.

The crystal must be of the fundamental frequency type. The frequency will be divided by four in order to create the internal clock frequency.

Both the TMS 9980A and the TMS 9981 output $\overline{\Phi 3}$, a synchronizing clock signal. $\overline{\Phi 3}$ is the inverse of the Φ3 clock signal shown in Figure 18-3 and in subsequent timing diagrams for the TMS 9900.

Thus you can create the timing diagram for any TMS 9980 operation by looking at the equivalent timing diagram for the TMS 9900 and replacing the four TMS 9900 clock signals by a single timing pulse which will be the complement of Φ3.

The following operations are identical within TMS 9900 and TMS 9980 systems:

- Memory references. However, note that memory reference will consist of two memory access cycles, as a 16-bit word is handled as two bytes.
- CRU I/O operations (remember that the TMS 9980 series CRU is only 2048 bits wide).
- CRU control operations
- The Wait state
- The Hold state and direct memory access operations
- The Halt state
- The interaction of Hold and Halt states

Refer to the TMS 9900 discussion for any of the above topics.

TMS 9980 SERIES INTERRUPT LOGIC

The TMS 9980A and TMS 9981 microprocessors support four levels of external interrupt, together with a Reset and a Load. Reset and Load are non-maskable interrupts. In contrast, the TMS 9900 supports 15 levels of external interrupt, along with Reset.

The TMS 9980 series microprocessors identify external interrupts via the INT0, INT1, and INT2 inputs as shown in Table 18-6. Figure 18-17 shows the interrupt vector map.

Table 18-6. TMS 9980 Interrupts

INT0	INT1	INT2	Interrupt Decoded
0	0	0	Reset
0	0	1	Reset
0	1	0	Load
0	1	1	Level 1 (Highest Priority)
1	0	0	Level 2
1	0	1	Level 3
1	1	0	Level 4 (Lowest Priority)
1	1	1	No Interrupts

Observe that the TMS 9980A and the TMS 9981 have no $\overline{\text{INTREQ}}$ input. Also, the Reset and Load non-maskable interrupts are decoded from the INT0 - INT2 inputs.

Figure 18-18 shows some pin connections for various levels of interrupt complexity in a TMS 9980 series microcomputer system. The three illustrations shown are self-evident; they simply implement the INT0 - INT2 codes defined above.

The TMS 9980 series microprocessors provide all 16 XOP software interrupts available with a TMS 9900.

Observe that Figure 18-17 shows memory as 8-bit units in contrast to Figure 18-11, which shows memory as 16-bit units. This reflects the fact that external memory is addressed as bytes by the TMS 9980A and the TMS 9981.

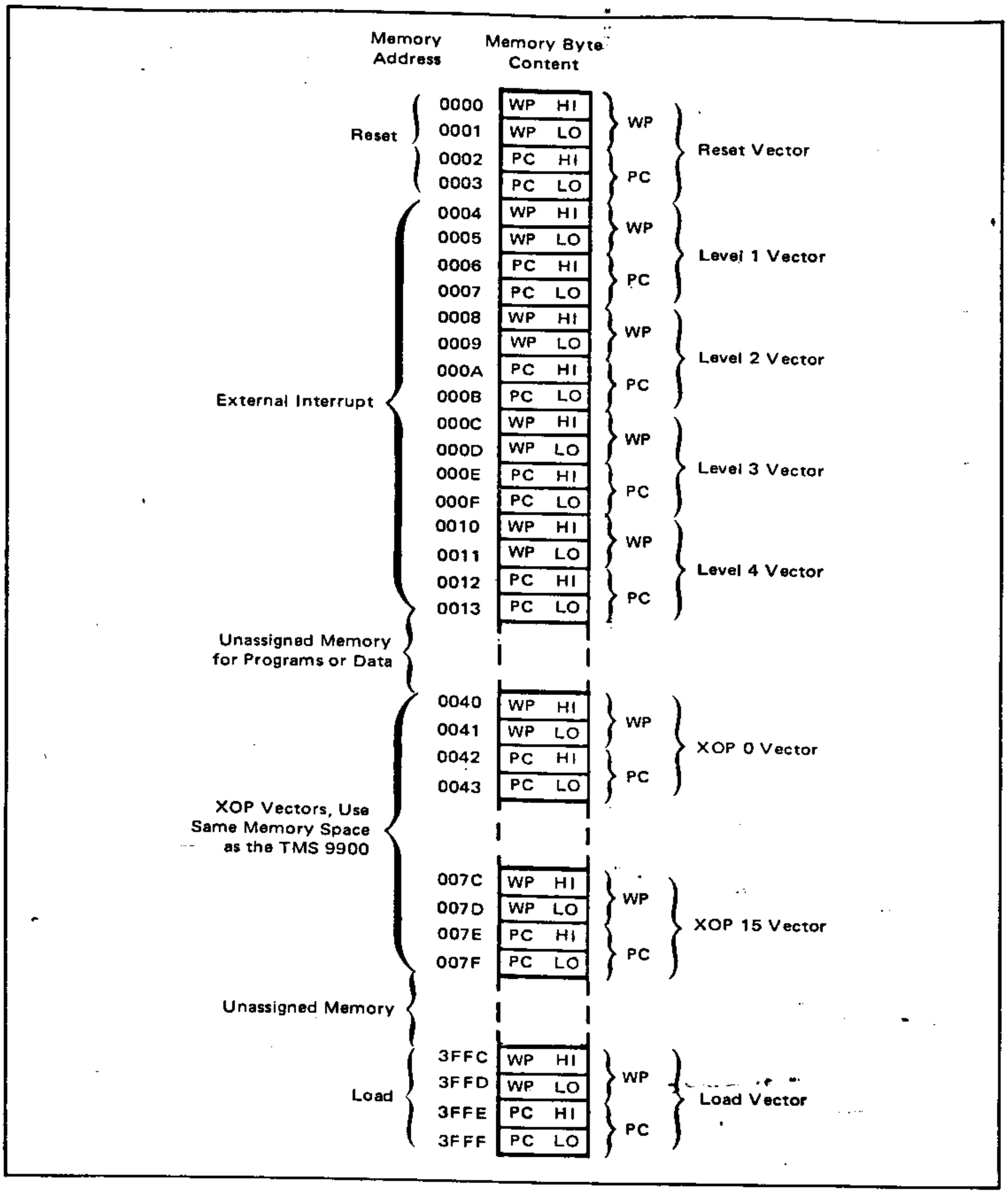


Figure 18-17. TMS 9980 Memory Map

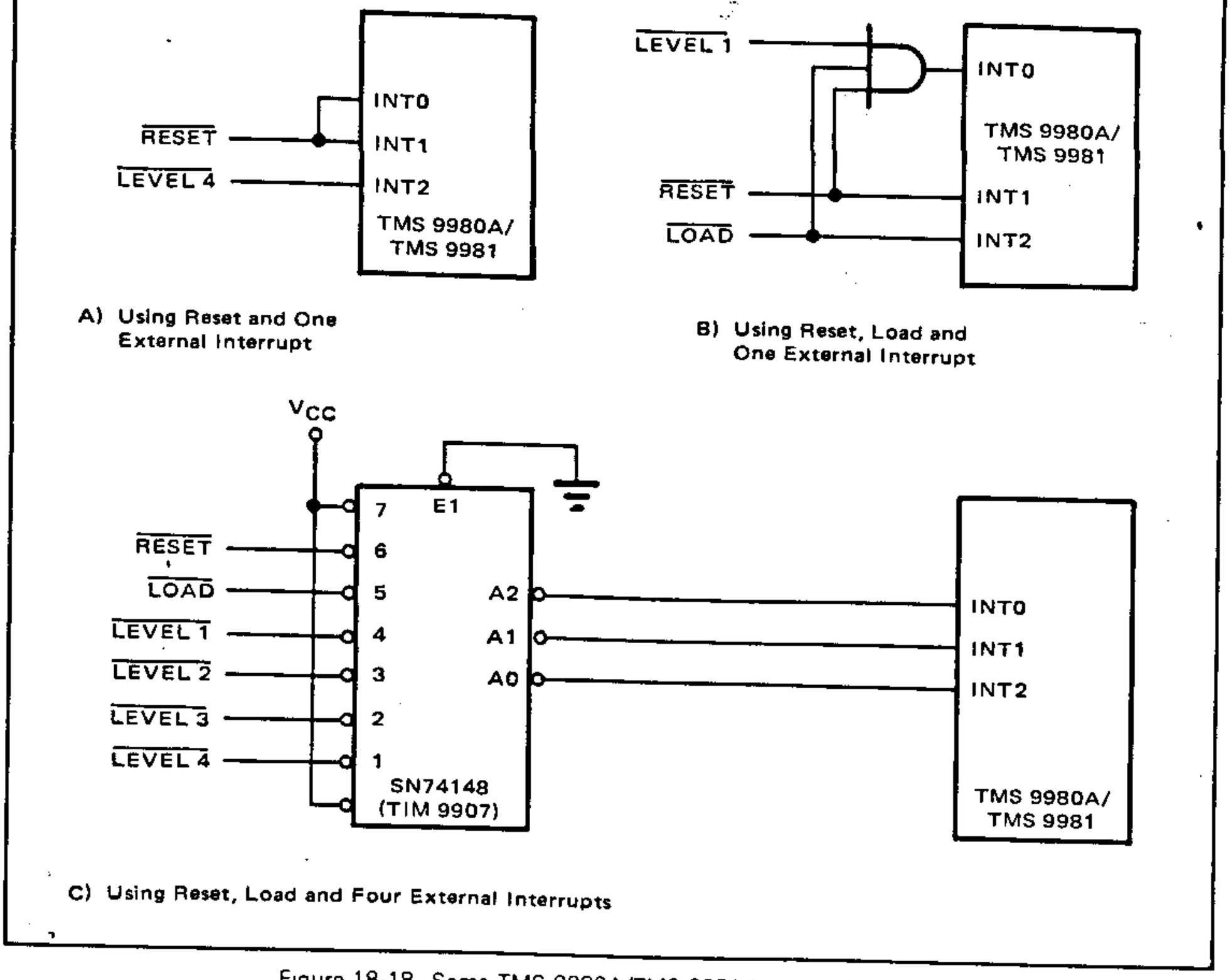


Figure 18-18. Some TMS 9980A/TMS 9981 Interrupt Interfaces

The interrupt acknowledge process and interrupt priority arbitration logic are identical in TMS 9900 and TMS 9980 series microprocessors. For a discussion of these subjects. refer to the earlier TMS 9900 description.

THE TMS 9980 SERIES INSTRUCTION SET

The TMS 9900 and TMS 9980 series microprocessors have identical instruction sets Instructions execute in almost the same sequences of machine cycles — the only difference is that each memory reference will have twice as many memory access cycles. Refer to Tables 18-2 and 18-3. together with their accompanying text. for details. Remember to substitute two memory cycles for each TMS 9900 memory cycle.

THE TMS 9940 SINGLE-CHIP MICROCOMPUTERS

The TMS 9940 is a single-chip microcomputer based on the TMS 9900 microprocessor. Figure 18-19 illustrates that part of our general microcomputer system logic provided by the TMS 9940 series microcomputer.

Specifically, this is the logic provided by the TMS 9940 series microcomputers:

- A Central Processing Unit, essentially equivalent to **the TMS 9900 Central Processing Unit**
- **2048 bytes of read-only memory.** Erasable Programmable Read-Only Memory (EPROM) is provided by the TMS 9940OE. Normal mask programmable Read-Only Memory (ROM) is available with the TMS 9940M.
- **128 bytes of read/write memory.** This read-write memory is frequently organized as four sets of sixteen 16-bit registers.

- **Two levels of external interrupt**
- **An on-chip timer/event counter with its own interrupt logic**
- **32 I/O pins accessed as 32 CRU bits**
- **A single +5V power supply**
- **On-chip clock logic**

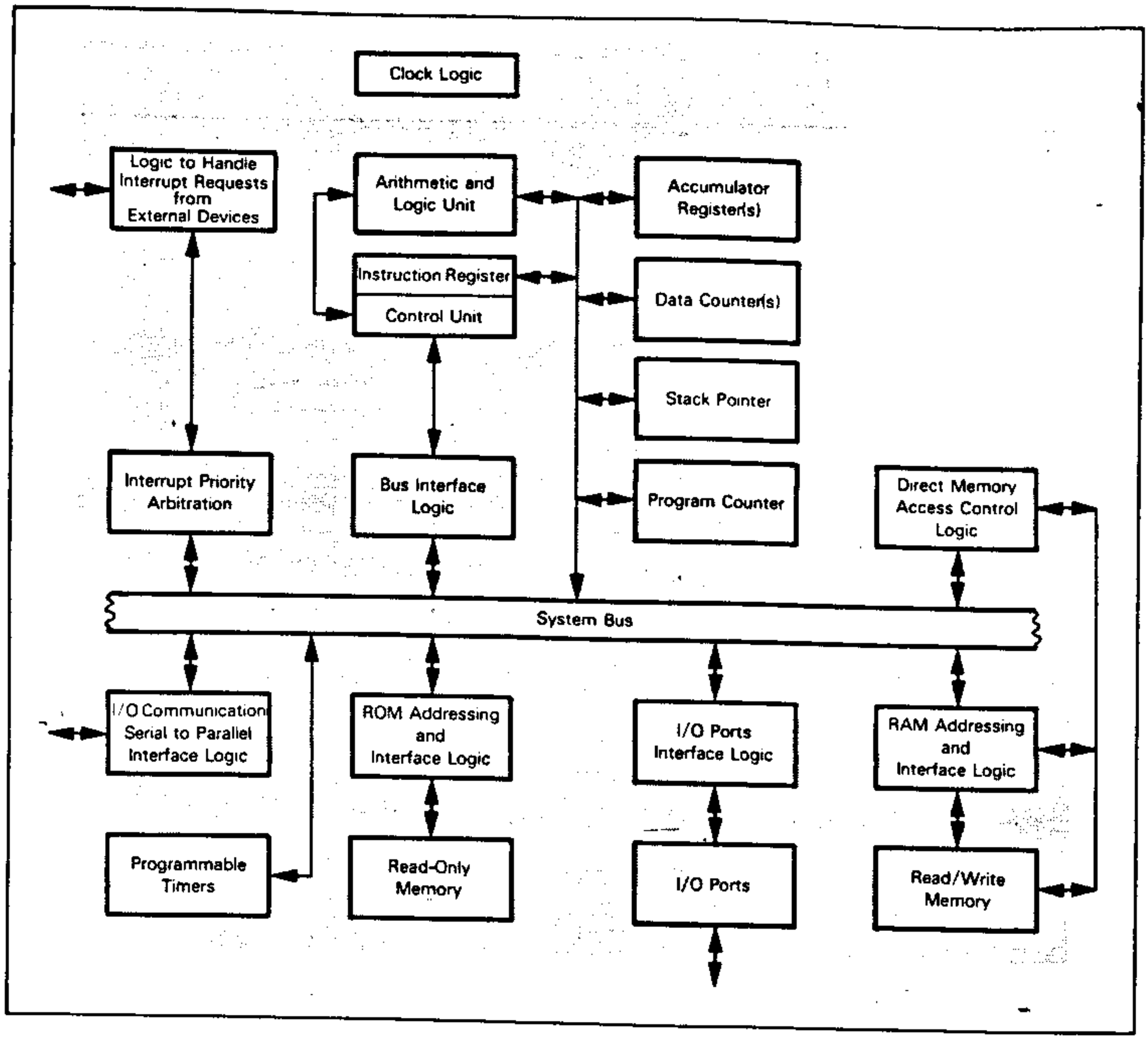


Figure 18-19. Logic of the TMS 9940 Single-Chip Microcomputers

The TMS 9940 microcomputer has very little expansion logic; 256 external CRU bits can be addressed, but there is no provision for executing programs directly from external memory.

But the TMS 9940 is easily included in multiprocessor configurations. For multiprocessor configurations, the TMS 9940 has internal Hold request/acknowledge logic, together with a serial I/O path via which data can be transferred between processors.

The TMS 9940 has two +5V power supplies: a standard operating power supply and a standby power supply. Under program control, it is possible to shut down the TMS 9940, in which case only the standby power supply is active. An external interrupt can subsequently restart the TMS 9940.

The TMS 9940 is manufactured using N-channel silicon gate MOS technology. It is packaged as a 40-pin DIP.

Using a 3 MHz clock, instruction execution times range between 3 and 10 microseconds.

This description of the TMS 9940 microcomputer relies on the preceding detailed description of the TMS 9900. This description of the TMS 9940 does not stand alone, and you should not read it until you understand the TMS 9900 in detail.

TMS 9940 REGISTERS AND READ/WRITE MEMORY

There are some important conceptual differences between the read/write memory/registers of the TMS 9940 and those of the TMS 9900.

The TMS 9940 has only 128 bytes of read/write memory, with all the read/write on the chip itself, and you cannot create an external Data/Address Bus. Therefore, it makes no difference whether memory is addressed as bytes or words. The only remaining restriction is that 16-bit words must be origined on even byte address boundaries.

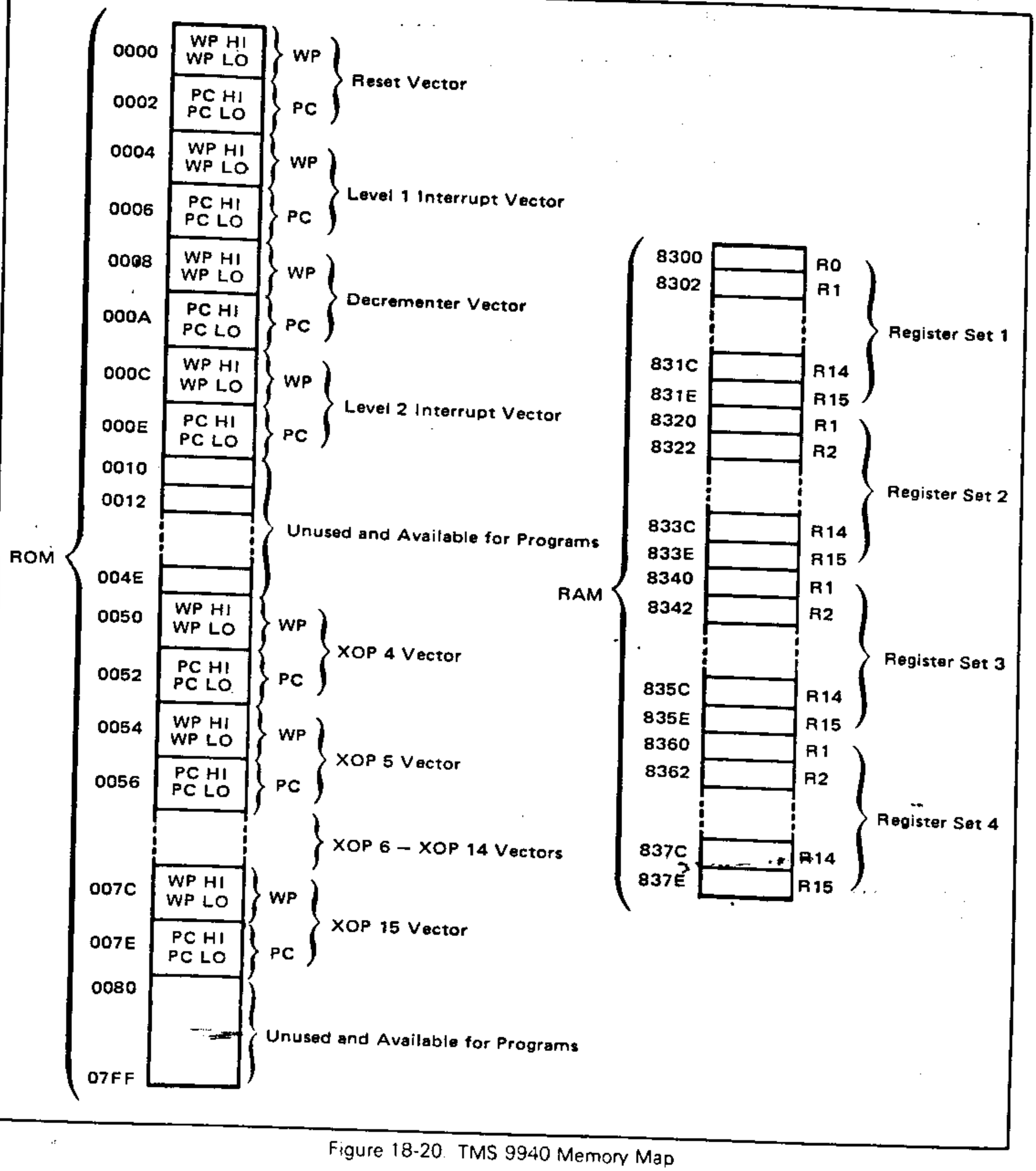


Figure 18-20. TMS 9940 Memory Map

The TMS 9940 does introduce one additional read/write memory restriction: the **128 bytes of read/write memory are divided into four non-overlapping sets of sixteen 16-bit registers, as illustrated in Figure 18-20.** Note that the 128 bytes of read/write memory have specifically defined addresses. Both the TMS 9900 and the TMS 9980 series microprocessors allow any sixteen 16-bit words of memory to serve as a set of general purpose registers. whether or not they overlap with another set.

The TMS 9940 has the same three CPU registers as the TMS 9900: the Program Counter. the Workspace register. and the Status register. The TMS 9940 sets aside general-purpose registers to serve specific functions. as does the TMS 9900.

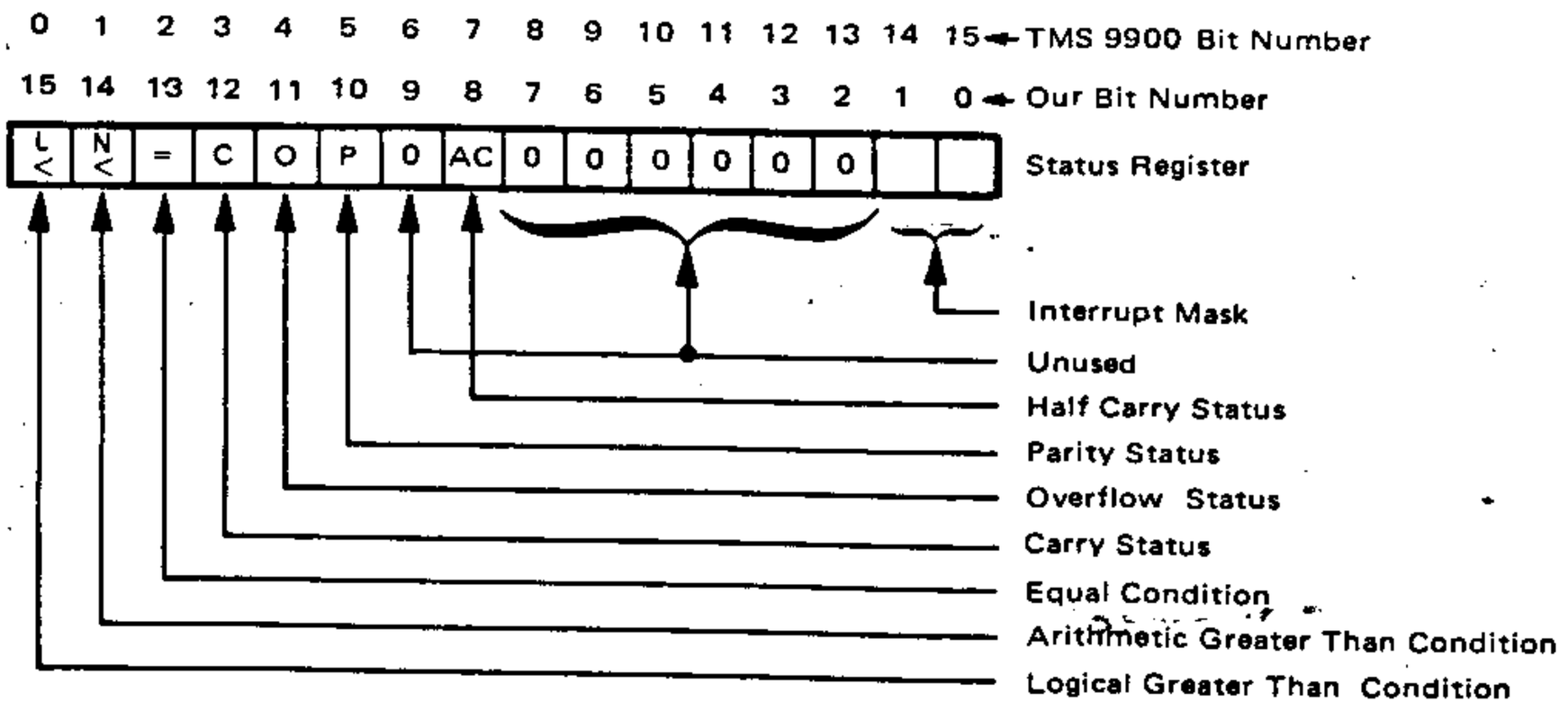
Given the configuration of the TMS 9940. many register designations can be justified only as a means of preserving TMS 9900 series compatibility. For example. a 16-bit TMS 9940 Workspace register makes no sense when there are only 64 locations that the Workspace register can possibly address. Moreover. the whole idea of context switching — and tying up three 16-bit registers in order to execute a context switch — is ridiculous. given the few places to which you can context switch.

But there is long-range sense in the TMS 9940 design. Over the next few years. enhancements of the TMS 9940 will appear with substantially more memory — both read-only memory and read/write memory. Since it is absolutely imperative that TMS 9940 programs be compatible with new. enhanced one-chip microcomputers that are likely to appear. it is necessary that addressing modes and architectural features that influence the instruction set be included in the TMS 9940 if they will be useful in later enhancements.

Despite the fact that the TMS 9940 has only 128 bytes of read/write memory and 2048 bytes of read-only memory. the TMS 9940 has all of the TMS 9900 memory addressing modes. Note carefully that so far as memory addressing is concerned. there is no difference between read-only memory and read/write memory. Many one-chip microcomputers have a scratchpad read/write memory which can only be accessed as data memory. while a separate program memory can only store instruction sequences. the TMS 9940 makes no such distinction between its read-only memory and read/write memory. Data and instructions can be stored in read-only memory or in read/write memory.

The TMS 9940 and TMS 9900 CRU addressing techniques are identical; however. the TMS 9940 has just 32 external CRU bits. each with its own dedicated pin. By configuring 11 of these pins as address lines and CRU controls. you can expand external CRU to 256 bits.

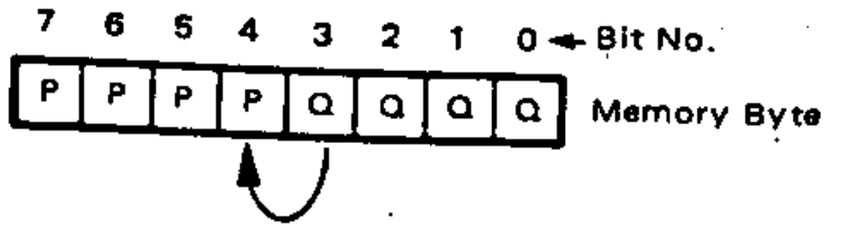
There are some small differences between the TMS 9930 Status register as compared to the TMS 9900 Status register. The TMS 9940 Status register may be illustrated as follows:

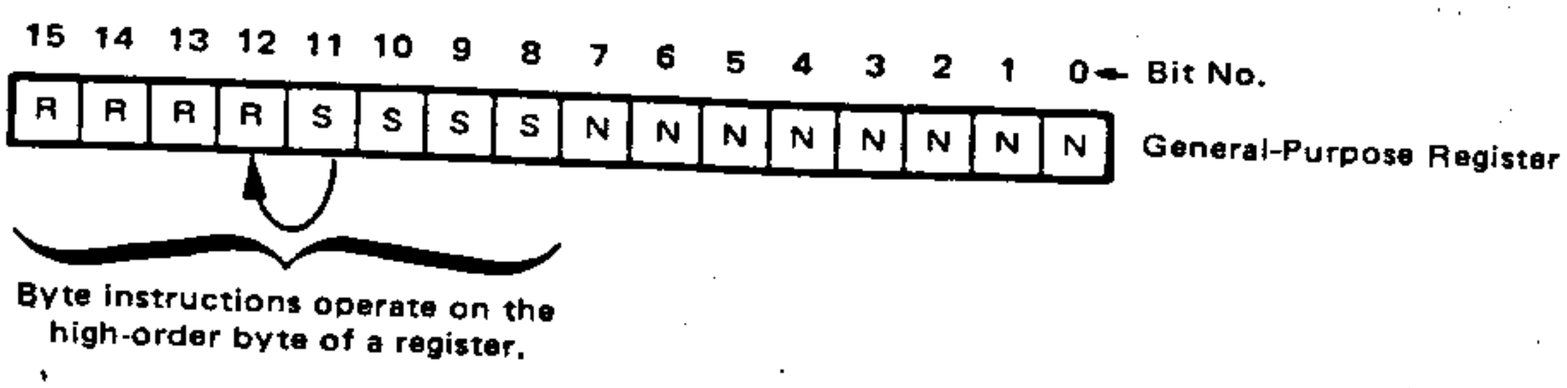


TMS 9940 L. N. =. C. O. and P statuses are the same as those of the TMS 9900.

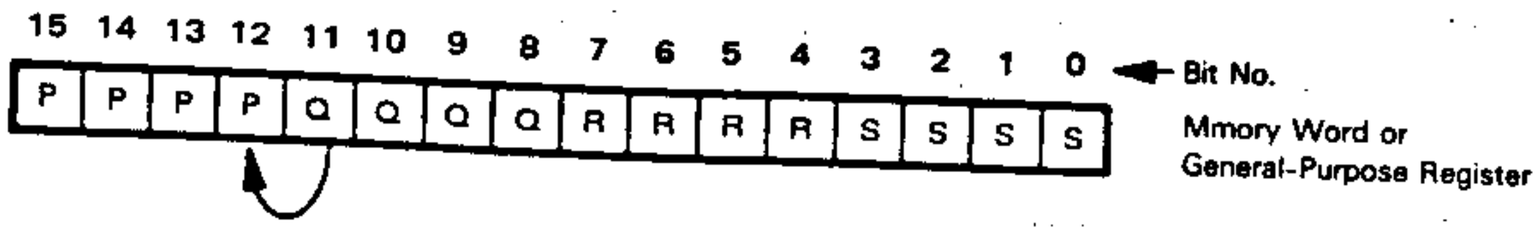
The TMS 9940 has no XOP instruction executed status. which the TMS 9900 holds in Status register bit 9.

The TMS 9940 has an AC status in bit 8. This is a half-carry status. For byte-oriented instructions. AC represents the carry from the low four bits to the higher four:





For 16-bit instructions. the AC status represents a carry from bit 11 to bit 12:



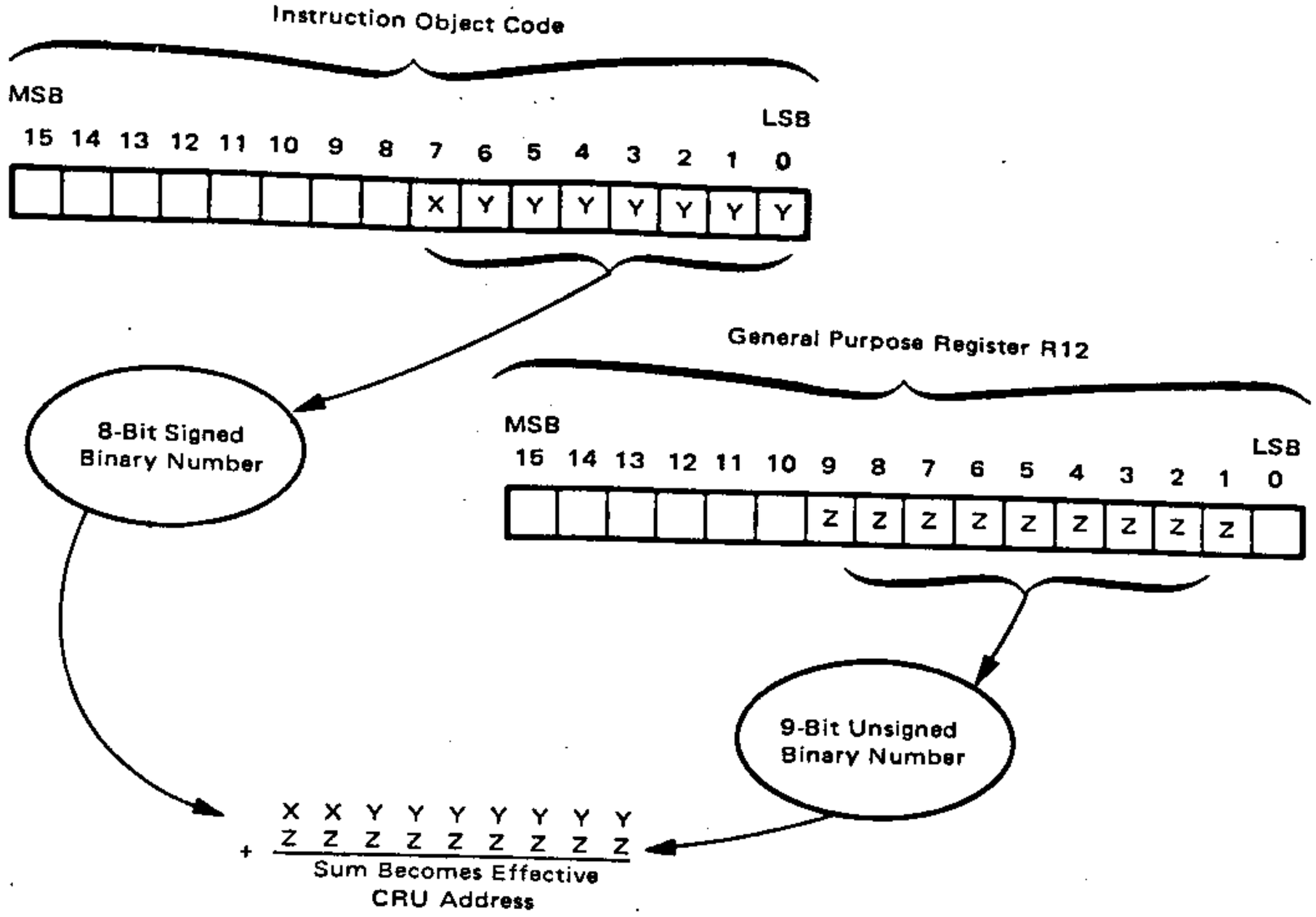
Since there are just four levels of external interrupt. the TMS 9940 uses Status register bits 0 and 1 for its interrupt mask. In contrast. the TMS 9900 uses Status register bits 0. 1. 2. and 3 for its interrupt mask.

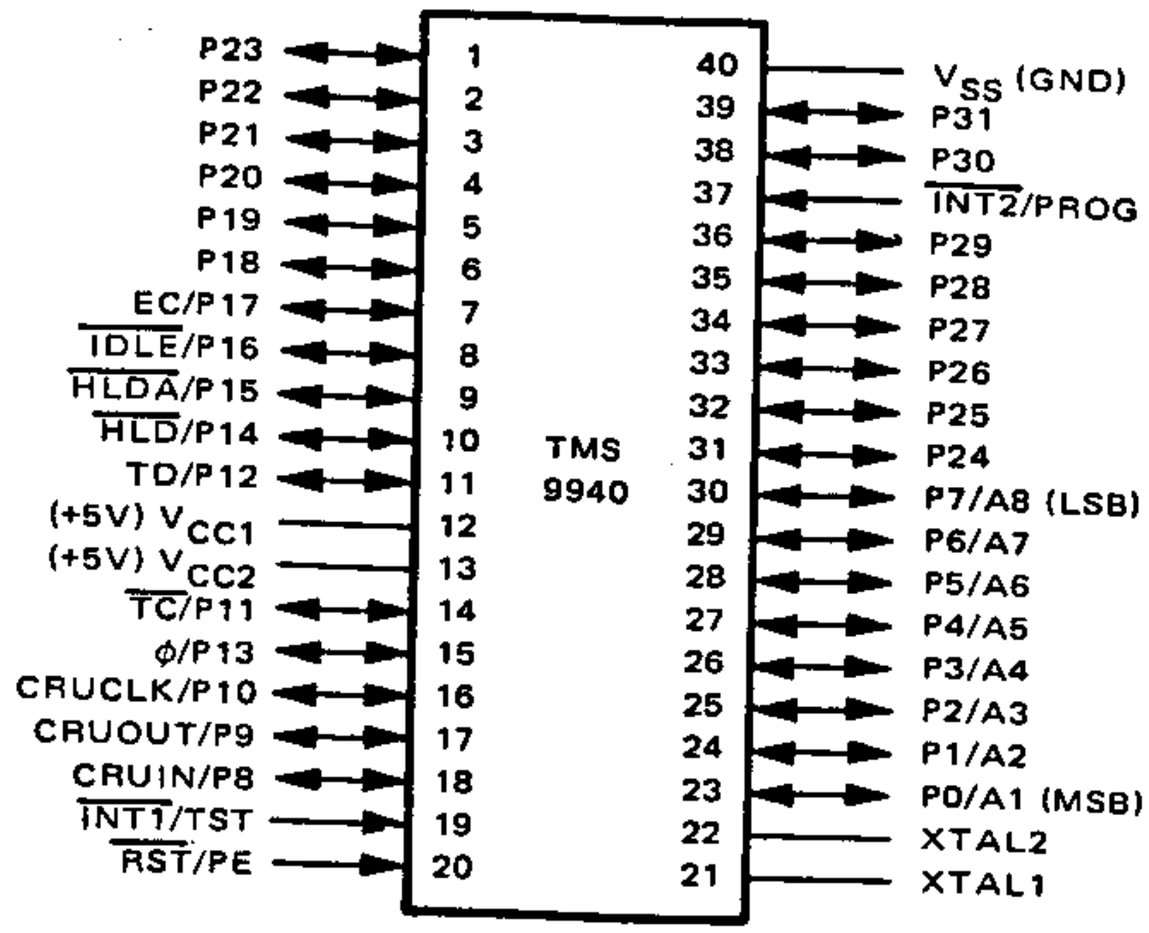
TMS 9940 CPU PINS AND SIGNAL ASSIGNMENTS

Figure 18-21 illustrates the pins and signals of the TMS 9940 microcomputer.

P0 - P31 and 32 I/O pins addressed as 32 CRU bits. Some of these pins serve additional functions which can be selected under program control.

The TMS 9940 can, in fact, use standard TMS 9900 CRU instructions to **address up to 512 CRU bits.** But 512 is the maximum number of CRU bits that the TMS 9940 can address. Therefore, the TMS 9940 uses just 9 bits of General Purpose Register R12 to create CRU bit addresses. For a single-bit CRU instruction, this may be illustrated as follows:





Pin Name	Description	Type
P0 - P31	CRU I/O pins	Bidirectional
INT1/TST	External interrupt and Test select	Input
INT2/PROG	External interrupt and EPROM programmer	Input
RST/PE	System reset and EPROM programmer enable	Input
A0 - A7	External CRU bit address	Output
CRUCLK	External CRU clock	Output
CRUOUT	External serial I/O output	Output
CRUIN	External serial I/O input	Input
TC	Multiprocessor data I/O clock	Bidirectional
TD	Multiprocessor data I/O	Bidirectional
EC	Event counter input	Input
IDLE	Idle state indicator	Output
HLD	Hold request	Input
HLDA	Hold acknowledge	Output
Φ	Synchronizing clock	Output
XTAL2, XTAL1	External crystal connections	
V_{CC1}	Standby +5V power	
V_{CC2}	Normal +5V power	
V_{SS}	Ground reference	

(In this figure, Pn and An numbering conforms to Texas Instruments' policy of beginning with N=0 for the high-order bit. We use N=0 for the low-order bit.)

Figure 18-21. TMS 9940 Microcomputer Signals and Pin Assignments

Table 18-7 shows how the TMS 9940 interprets its 512 available bit addresses.

Table 18-7. TMS 9940 CRU Bit Address Assignments

CRU Address	Read Function	Write Function
000 to 0FF	External CRU bits; the address is output via A1-A8. Data is transferred via CRUIN, CRUOUT and CRUCLK	
100 to 17F	Unused	Unused
180	INT1 state	Unused
181	Decrementer interrupt level	Clear decrementer interrupt
182	INT2 state	Unused
183	Unused	Configuration bit 0 (CB0)
184	Unused	Configuration bit 1 (CB1)
185	Unused	Configuration bit 2 (CB2)
186	Unused	Configuration bit 3 (CB3)
190 to 19D	Decrementer register. 190 is the least significant bit and 19D is the most significant bit	
19E	Unused	Timer (high) or Counter (low) select
19F	Unused	Unused
1A0 to 1AF	Multiprocessor System Interface buffer register 1A0 is the least significant bit and 1AF is the most significant bit	
1B0 to 1BF	General purpose flag bits	
1C0 to 1DF	Unused	Identify direction for P0 (via 1C0) through P31 (via 1DF). 1 specifies output. 0 specifies input
1E0 to 1FF	Local CRU pins (P0 = 1E0, P31 = 1FF	

TMS 9940 CRU BIT UTILIZATION

The place to begin looking at Table 18-7 is at CRU bits 183, 184, 185, and 186. These four CRU bits represent write-only locations which determine how the 32 CRU pins illustrated in Figure 18-21 will be used.

If you look again at Figure 18-21, you will see that P0 through P17 have shared functions. P18 through P31 are simple I/O pins without other programmable options.

CRU addresses 183, 184, 185 and 186 control the functions of P0 through P16, as illustrated in Table 18-8. P17 options depend on real-time clock logic, which we will describe later.

Let us look at the programmable options available with CRU pins P0 through P31.

It does not matter what options you have selected: **you will actually access the 32 CRU pins P0 - P31 via CRU addresses $1E0_{16}$ through $1FF_{16}$.**

TMS 9940 SIMPLE CRU I/O MODE

In the simplest case, all 32 pins, P0 - P31, will be used for input or output. We call this Simple I/O mode. In order to use all 32 pins for data input or output, (that is, in Simple I/O mode), all four of the configuration bits, CB0, CB1, CB2, and CB3, must be 0. At any time, a CRU bit can either input data or output data, but it cannot be used for bidirectional data transfer. **You must identify the direction for each pin by outputting appropriate data to CRU addresses $1C0_{16}$ through $1DF_{16}$.** As shown in Table 18-7, each pin has a dedicated CRU address, beginning with pin P0 controlled by

$1C0_{16}$ and ending with pin P31 controlled by CRU address $1DF_{16}$. A 1 written to any Direction CRU bit causes the associated pin to output data only. A 0 written to any CRU Direction bit causes the associated pin to input data only. Of course, you can at any time change a pin from input to output or from output to input, under program control, by rewriting control information to Direction CRU bits $1C0_{16}$ through $1DF_{16}$

Table 18-8. TMS 9940 CRU Bits Whose Functions are Determined Under Program Control

CRU			Function as Configured		
Bit	Address	Pin	CB0 = 0	CB0 = 1	CB1, CB2, CB3
0-7	1E0-1E7	23-30	P0-P7	A1-A8	No Effect
8	1E8	18	P8	CRUIN	No Effect
9	1E9	17	P9	CRUOUT	No Effect
10	1EA	16	P10	CRUCLK	No Effect
			CB1 = 0	CB1 = 1	CB0, CB2, CB3
11	1EB	14	P11	$\overline{TC}$	No Effect
12	1EC	11	P12	TD	No Effect
			CB2 = 0	CB2 = 1	CB0, CB1, CB3
13	1ED	15	P13	ϕ	No Effect
			CB3 = 0	CB3 = 1	CB0, CB1, CB2
14	1EE	10	P14	$\overline{HLD}$	No Effect
15	1EF	9	P15	$\overline{HLDA}$	No Effect
16	1F0	8	P16	$\overline{IDLE}$	No Effect

You will always have to define the direction of data transfer for pins P18 through P31 — assuming that you are using these pins. When pins P0 through P17 are being used in any of the special ways which we are about to describe, then the data direction associated with the special operation will apply, and it makes no difference what you output to the associated Direction CRU bit.

TMS 9940 CRU I/O EXPANSION MODE

If you wish to use 256 external CRU bits, then you must set CRU bit 183 (CB0) to 1. This is called I/O expansion mode. I/O expansion mode modifies the functions of pins P0 through P10. When you use CRU addresses 00 through FF_{16} in I/O expansion mode, the address is output via pins P0 - P7, which now function as CRU address lines A1 - A8. P8, P9, and P10 serve as the standard CRU data transfer lines: CRUIN, CRUOUT, and CRUCLK. Timing for data input and output via these three lines has been described for the TMS 9900. Refer to the TMS 9900 description for details. **In order to illustrate the use of external CRU, consider execution of the instructions:**

```
LI      R3.>00      LOAD 1010 BINARY INTO UPPER BYTE OF R3
LI      R12.>140    LOAD A BASE ADDRESS OF 82 HEX INTO R12
LDCR    R3.4        OUTPUT FOUR LOW-ORDER BITS OF R3 TO CRU
```

Note that R12 contains 0140_{16} to represent the address 082_{16}, since R12 bit 0 is unused; therefore the internal address is, in effect, doubled.

This instruction outputs 1010 to CRU bit 082_{16} (0), 083_{16} (1), 084_{16} (0), and 085_{16}(1). Since fewer than eight bits will be transferred, they will come from the upper byte of the general purpose register. This is the event sequence which occurs:

1) The address 82_{16} is output via A1 - A8. Remember, Texas Instruments' literature uses 0 to represent the high-order bit; therefore A1 represents the high-order address bit, and A8 represents the low-order address bit. CRUIN is inactive, but CRUOUT is low to represent 0 while CRUCLK is pulsed high to time the 0 bit on CRUOUT.

2) The address output on A1 - A8 increments to 83_{16}, and CRUOUT goes high, then CRUCLK pulses high.

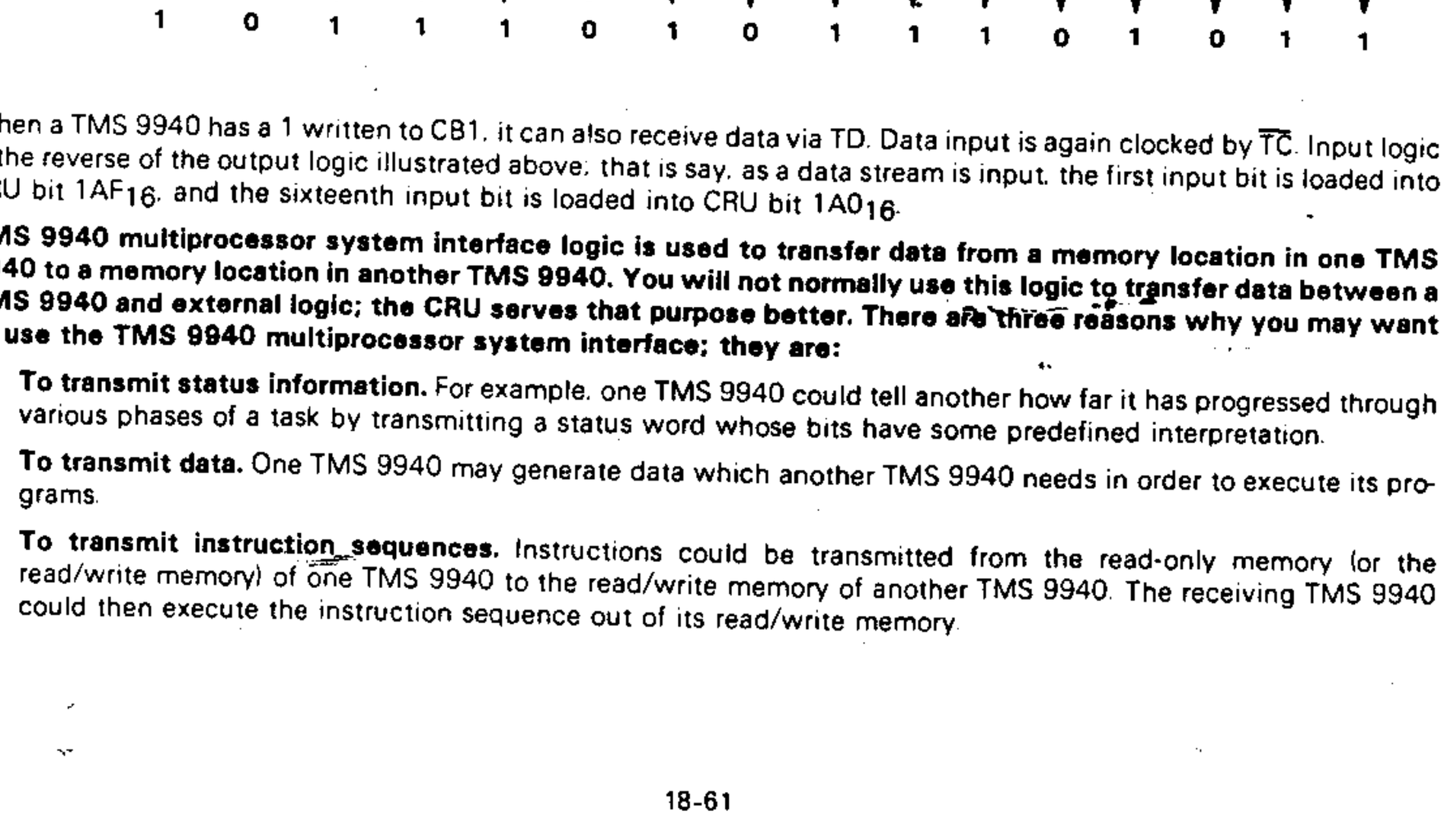
3) The address on A1 - A8 increments to 84_{16}. CRUOUT goes low again, and CRUCLK pulses high.

4) The address on A1 - A8 increments to 85_{16}, and CRUOUT goes high, and CRUCLK pulses high.

1010 has now been transmitted to four external CRU bits.

Note that it is up to external logic to decode the CRU address output; however, the Parallel System interface (which we will describe in later editions) will connect directly to the TMS 9940 Address and CRU outputs that we have just described.

When you write 1 to CRU bit 184_{16} (CB1), pins P11 and P12 function as serial data transfer pins. The purpose of this logic is to allow the TMS 9940 to operate in multi-CPU configurations. This logic is very simple. You output data by writing the data to CRU bits $1A0_{16}$ through $1AF_{16}$. This data is immediately transmitted via TD (P12) as a serial data stream which is clocked by $\overline{TC}$ (P11). In keeping with normal bit sequence protocol, data is transmitted low-order bit first. Thus, 16 bits of data being output may be illustrated as follows:



When a TMS 9940 has a 1 written to CB1, it can also receive data via TD. Data input is again clocked by $\overline{TC}$. Input logic is the reverse of the output logic illustrated above; that is say, as a data stream is input, the first input bit is loaded into CRU bit $1AF_{16}$, and the sixteenth input bit is loaded into CRU bit $1A0_{16}$.

TMS 9940 multiprocessor system interface logic is used to transfer data from a memory location in one TMS 9940 to a memory location in another TMS 9940. You will not normally use this logic to transfer data between a TMS 9940 and external logic; the CRU serves that purpose better. There are three reasons why you may want to use the TMS 9940 multiprocessor system interface; they are:

1) **To transmit status information.** For example, one TMS 9940 could tell another how far it has progressed through various phases of a task by transmitting a status word whose bits have some predefined interpretation.

2) **To transmit data.** One TMS 9940 may generate data which another TMS 9940 needs in order to execute its programs.

3) **To transmit instruction sequences.** Instructions could be transmitted from the read-only memory (or the read/write memory) of one TMS 9940 to the read/write memory of another TMS 9940. The receiving TMS 9940 could then execute the instruction sequence out of its read/write memory.

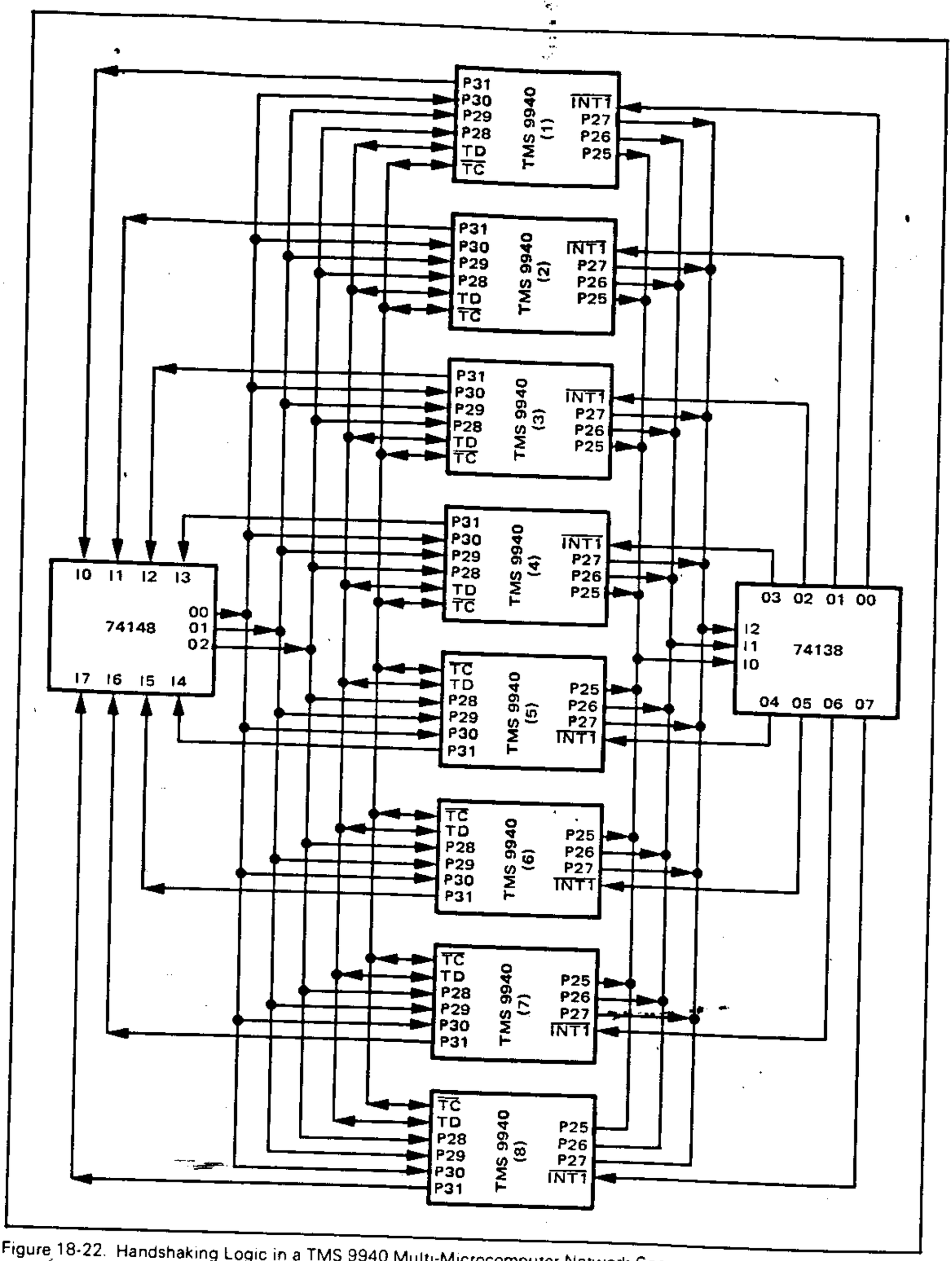


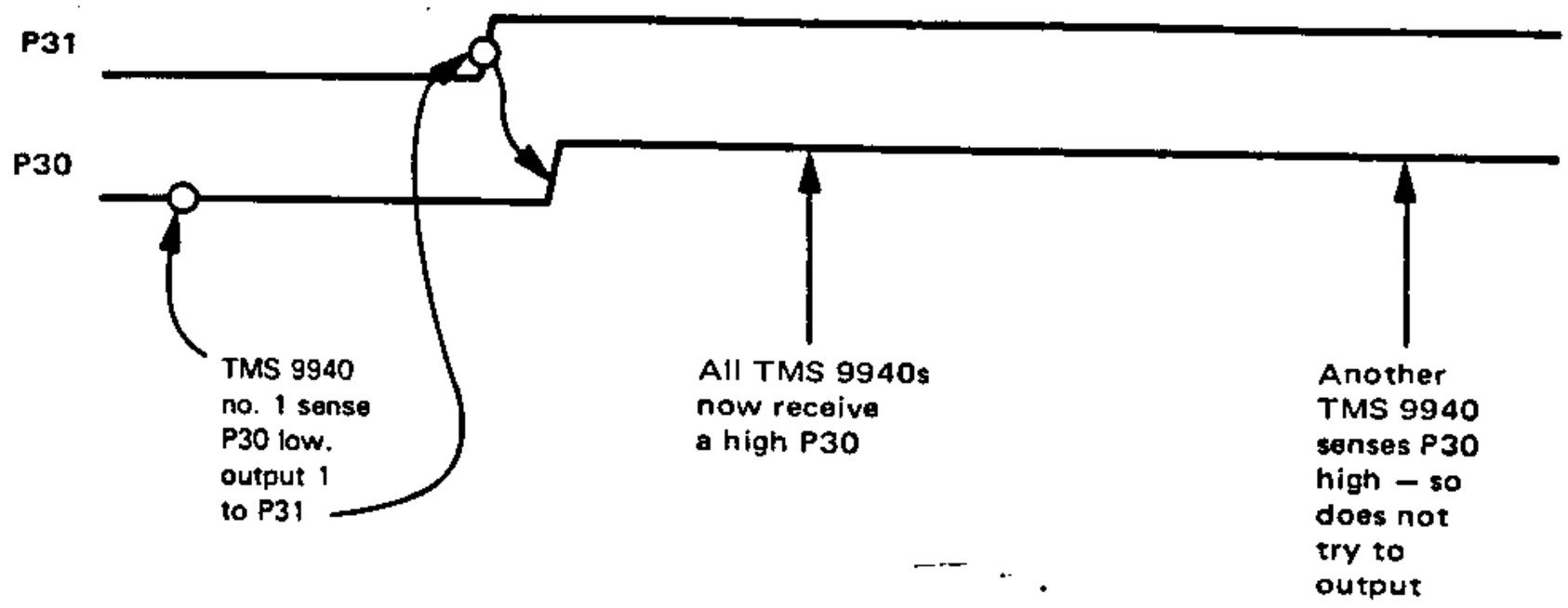
Figure 18-22. Handshaking Logic in a TMS 9940 Multi-Microcomputer Network Communicating via the TD Data Line

You could use the CRU to perform any of the three data transfers described above. but the multiprocessor system interface is somewhat easier to use. We say that data transfer via the multiprocessor system interface is "somewhat" easier to use because many problems still remain when you use the multiprocessor system interface. These problems arise from the fact that **there is absolutely no handshaking protocol associated with the multiprocessor system interface.** For example. there is absolutely no protection against two TMS 9940s simultaneously trying to output data via TD and $\overline{TC}$. There is no predefined protocol whereby a transmitting TMS 9940 identifies the receiving TMS 9940 or the instant data has been transmitted and should be read. Any protocol is your responsibility — to be provided by logic external to the TMS 9940s. Fortunately. **this protocol is easy to implement. Figure 18-22 shows how eight TMS 9940s can communicate with each other, such that each TMS 9940 may transmit data to, or receive data from, any other TMS 9940.** The logic illustrated in Figure 18-22 is more complex than the logic you would need for a small system — for example. a two-microcomputer system. or a system where there are dedicated transmitters and receivers.

While Figure 18-22 shows TMS 9940s communicating with each other. you will in fact use TMS 9940s just as frequently with other microprocessors — such as a TMS 9900. Nevertheless. the concepts embodied in Figure 18-22 would apply. from the viewpoint of the TMS 9940. in any other configuration.

Let us look at how the logic in Figure 18-22 works.

The first problem we must resolve is the problem of transmission contentions. How will we make sure that one TMS 9940 does not try to transmit data while another TMS 9940 is already transmitting data? A simple scheme would be to set aside a particular CRU pin to serve as a "Busy" line. For example. every TMS 9940 could use P31 as a "Busy" output pin and P30 as a "Sense" input pin. We could wire-OR together all P31 Busy outputs and input this wire-OR to all P30 Sense inputs. Now any TMS 9940 that wishes to transmit data will read its P30 CRU bit. If this bit is 0. then it will output 1 to P31. Outputting 1 to P31 causes all other TMS 9940s to receive 1 at their P30 inputs. Thus. no other TMS 9940 will begin transmitting data if another TMS 9940 was in the process of transmitting data. This logic may be illustrated as follows:



The problem with the logic illustrated above is that two TMS 9940s could simultaneously read P30. find it was 0. output 1 to P31. then output competing data on TD. While the chances of two microcomputers executing identical instructions at exactly the same time are very small. a well-designed microcomputer system must account for every potential error. In Figure 18-22 we resolve our problem by using a 74148 8-to-3 decoder. The P31 output from every TMS 9940 is connected to a different 74148 input. The 74148 outputs. via O0. O1. and O2. the line number for the highest priority active input. This three-line output is connected to the P28. P29. and P30 pins of every TMS 9940; we assume that these three pins are inputs at every TMS 9940. Now every TMS 9940 that wishes to transmit data via TD must output a 1 to P31. It must then input the contents of P30. P29. and P28. Upon detecting its own ID on these three inputs. it begins data transmission. If a TMS 9940 outputs 1 via P31 and then reads in some other ID via P30. P29. and P28. then it must wait. Here is an appropriate instruction sequence:

```
        LI      R12.>3F8    LOAD P28 ADDRESS. X2. INTO R12
        SBO     3           SET P31 ON
LOOP    STCR    R2.3        INPUT P28. P29. AND P30
        CI      R2.ID       COMPARE INPUT WITH DEVICE ID
        JNE     LOOP        RETURN AND RE-ENTER CODE IF NOT CORRECT ID
        LI      R12.>340    LOAD MPSI OUTPUT DATA BASE ADDRESS X2
        LDCR    R3.16       OUTPUT CONTENTS OF R3 VIA TD
```

Assuming that a TMS 9940 has output 1 to P31 and has received back its own ID via P28, P29, and P30, the TMS 9940 is ready to transmit data. However, in addition to simply transmitting the data, the TMS 9940 must tell the intended recipient that the data has been transmitted. In Figure 18-22 we use a 74138 3-to-8 demultiplexer for this purpose. Pins P25, P26, and P27 of every TMS 9940 are outputs that connect to the I0, I1, and I2 inputs of the 74138. The transmitting TMS 9940 outputs data which will be received by every other TMS 9940; however, the transmitting TMS 9940 follows up by outputting a 3-bit code via P25, P26, and P27; this 3-bit code identifies the intended recipient. The 3-bit code is input to the 74138, which generates one of eight possible outputs. These eight outputs become external interrupt request inputs to the eight TMS 9940s. Only the single TMS 9940 will receive the data which was transmitted by the eighth TMS 9940, only one TMS 9940 will receive an interrupt request signal; this is the TMS 9940 for which the transmitted data was intended. The TMS 9940 which receives data simply executes an STRCR instruction to move the data from CRU bits $1A0_{16}$ through $1AF_{16}$ to the appropriate general purpose register.

TMS 9940 SYNC MODE

CRU bit 185_{16}, the CB2 bit, serves the very limited purpose of outputting a synchronizing signal. When you output 1 to CB2, P13 ceases to be an I/O pin and instead outputs the internal TMS 9940 clock signal.

CRU bit 186_{16} (CB3) controls idle and hold logic for the TMS 9940. When you write a 1 to CRU bit 186_{16}, pins P14 and P15 act as hold request input ($\overline{HLD}$) and hold acknowledge output ($\overline{HLDA}$) signals, respectively. P16 generates an $\overline{IDLE}$ output.

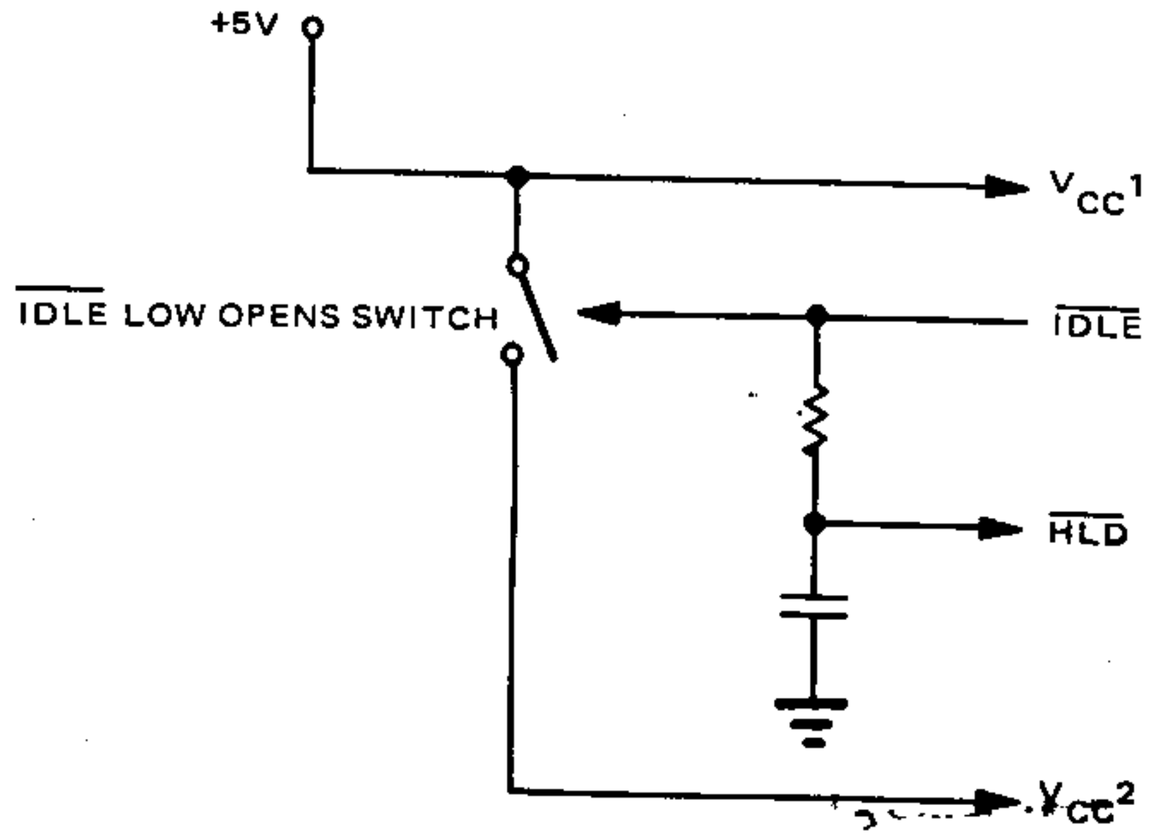
TMS 9940 HOLD LOGIC

The Hold request/acknowledge logic of the TMS 9940 is quite standard. The purpose of this logic is to remove the TMS 9940 from any shared busses when some other microprocessor or microcomputer is bus master. If CB3 is 1, then a low signal arriving at the TMS 9940 $\overline{HLD}$ input will cause the TMS 9940 to enter a Hold state at the conclusion of the current instruction's execution. A low $\overline{HLDA}$ output marks the beginning of the Hold state.

TMS 9940 IDLE LOGIC

The $\overline{IDLE}$ signal is output low when an IDLE instruction is executed and CB3 is 1. The only way in which you can terminate an Idle state is by requesting an interrupt via $\overline{INT1}$ or $\overline{INT2}$. The TMS 9940 three-state signals are not floated in the Idle state. You must additionally enter the Hold state for this.

The purpose of the IDLE instruction and signal is to enable standby power logic. This may be illustrated as follows:



Under normal circumstances, the power supply will input power to $V_{CC}1$ and $V_{CC}2$. When $\overline{IDLE}$ goes low, the power input to $V_{CC}2$ is switched off. While $V_{CC}1$ only is receiving power, the TMS 9940 read/write memory and interrupt logic is active, but all other logic is inactive since the interrupt logic is active, any arriving interrupt request will be acknowledged. The process of acknowledging an interrupt request sets $\overline{IDLE}$ high again. This closes the switch and restores power to $V_{CC}2$, which allows the TMS 9940 to resume normal execution.

In the illustration above, note that $\overline{IDLE}$ is connected to $\overline{HLD}$

TMS 9940 GENERAL PURPOSE FLAGS

If you look again at Table 18-7, you will see that CRU addresses $1B0_{16}$ through $1BF_{16}$ address 16 general purpose flags. These general purpose flags have no special hardware functions. They are programming aids and that is all. You can write data out to these flags, and you can read back the data. How you use this data is entirely up to program logic.

TMS 9940 TIMER/EVENT COUNTER LOGIC

The TMS 9940 has a timer which can also be used as an event counter. CRU bit $19E_{16}$ determines whether this logic will function as a timer or as an event counter. If CRU bit $19E_{16}$ is high, then this logic serves as a Timer, If CRU bit $19E_{16}$ is low, then this logic serves as an event counter.

Timer and Event Counter logic both use CRU bits 190_{16} through $19D_{16}$ as a 14-bit register whose contents are decremented by Timer or Event Counter logic. This 14-bit register is buffered. That is to say, the initial value which you output to CRU bits 190_{16} through $19D_{16}$ is stored in a buffer, in addition to being loaded into CRU bits 190_{16} through $19D_{16}$. Subsequently, CRU bits 190_{16} through $19D_{16}$ are decremented, but the buffer contents remain unaltered. When CRU bits 190_{16} through $19D_{16}$ decrement to 0, they are reloaded from the buffer. Thus Timer/Event Counter logic runs continuously. An interrupt request is generated internally when CRU bits 190_{16} through $19D_{16}$ decrement to 0.

Remember, CRU bit 190_{16} is the low-order bit, and CRU $19D_{16}$ is the high-order bit. This is the reverse of normal Texas Instruments bit numbering, where the high-order bit has the lowest bit number. However, this is consistent with the fact that Texas Instruments outputs data to the CRU low-order bit first, and addresses CRU bits in numerically ascending address sequence.

When you write 0 to CRU bits 190_{16} through $19D_{16}$, you disable Timer/Event Counter logic.

When the Timer/Event Counter is operating as a timer, the 14-bit register represented by CRU bits 190_{16} through $19D_{16}$ are decremented once every 30 internal clock oscillations. The crystal connected across XTAL1 and XTAL2 determines clock oscillation frequency. When CRU bits 190_{16} through $19D_{16}$ time out to zero, an interrupt request is generated.

When Timer/Event Counter logic is operating as an event counter, pin P17 serves as an input, receiving the event sequence to be counted. Every low-to-high transition of the signal input at P17 decrements the counter. Once again, when the counter counts out to 0, an interrupt request occurs and the counter is reloaded from its buffer register.

TMS 9940 INTERRUPT LOGIC

The TMS 9940 has four external interrupts and twelve internal software interrupts.

These are the four external interrupts:

1) Reset. This has highest priority.
2) A level 1 interrupt occurring at the $\overline{INT1}$ pin. This has second highest priority.
3) A Decrementer/Event Counter interrupt. This has third highest priority.
4) A level 2 interrupt occurring at the $\overline{INT2}$ pin. This has lowest priority.

As described for the TMS 9900, you execute XOP instructions to generate software interrupts. XOP4 through XOP15 are active. XOP0 through XOP3 do not exist on the TMS 9940.

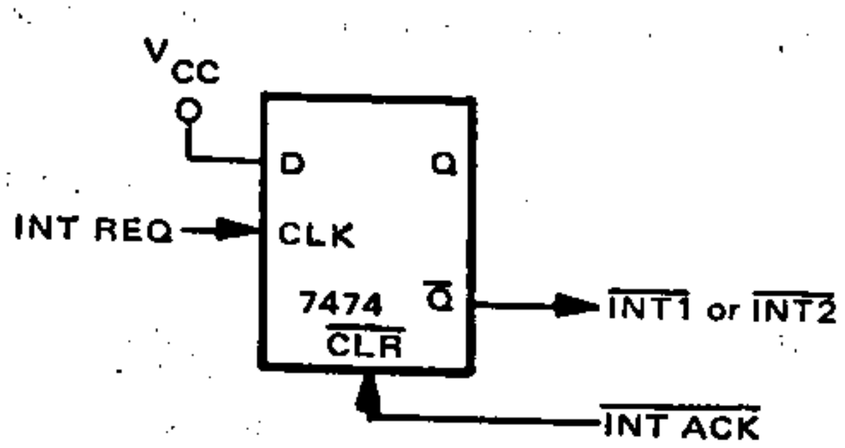
TMS 9940 interrupt vectors, together with a complete TMS 9940 memory map, are illustrated in Figure 18-20.

The actual interrupt acknowledge sequence for a TMS 9940 is identical to that which we have described for the TMS 9900.

TMS 9940 RESET

You Reset the TMS 9940 by inputting a low signal at $\overline{RST}$/PE (pin 20). This low signal must last for at least five clock cycles. A Reset resets to 0 the contents of all pointer registers and all CRU configuration bits. Following a Reset, level 0 interrupt response begins — which means that read-only memory bytes 0 through 3 provide the initial Program Counter and Word Pointer register contents, and therefore the address of the program which will be executed following the Reset.

Note that the TMS 9940. being a smaller and simpler system than the TMS 9900. can use elementary logic to generate an interrupt acknowledge. For the TMS 9900 we suggested an Address Bus decoding technique in order to create an interrupt acknowledge signal. For the TMS 9940 a CRU bit will do just fine. The following circuit is recommended by Texas Instruments:



A simple D-type flip-flop has its D input connected to +5V. Every time an interrupt request pulse is input to the clock pin. the $\overline{Q}$ output will go low — generating a valid interrupt request at the TMS 9940. In order to acknowledge the interrupt and remove the interrupt request signal. you can output a low pulse via any of the P pins. This low pulse clears the D-type flip-flop and forces $\overline{Q}$ high again.

PROGRAMMING A TMS 9940E ERASABLE, PROGRAMMABLE READ-ONLY MEMORY

The TMS 9940E has a transparent quartz lid over the device in its dual in-line package. **In order to erase the TMS 9940E EPROM, you should expose it to a high-intensity ultraviolet light with a wavelength of 2537 angstroms.** An intensity of 10 watt-seconds per square centimeter is recommended.

After the TMS 9940E EPROM has been erased. all EPROM memory bits will be 0.

These are the steps required in order to program a TMS 9940E EPROM:

1) Reset the device.
2) Apply the first data byte — to be stored in memory location 0000 to pins P24 through P31. Remember. P24 represents the most significant bit of the byte. and P31 represents the least significant bit of the byte.
3) Apply a 26-volt level to pin 20. the $\overline{RST}$/PE pin. This being the first programming pulse. it resets the internal program memory address point at 0000 and writes the data byte at P24 through P31 into memory location 0.
4) After at least 80 clock cycles. apply 26 volts to pin 37. $\overline{INT2}$/PROG. for 50 milliseconds while changing the data byte (step 5).
5) Apply the next data byte to P24 through P31. At the high-to-low transition of PROG. the data will be written into the next location.
6) Remove the 26 volts from pin 37 for a minimum of 50 clock cycles. Then apply 26V to pin 37 for 50 milliseconds.
7) Return to Step 5 until all of program memory has been programmed.

LOADING A PROGRAM INTO TMS 9940 READ/WRITE MEMORY

You can load a program directly into TMS 9940 read/write memory via pins P24 (MSB) through P31 (LSB) for either the TMS 9940E or the TMS 9940M. Typically. this is done in order to load a small test program. The procedure for loading data into the TMS 9940 read/write memory is exactly as described in the previous section for loading data into EPROM; except: the 26-volt level is applied to pin 19. the TST pin. after the device has been reset by inputting a low signal to pin 20. the $\overline{RST}$/PE pin; and the high pulses at PROG are logic '1' level rather than 26 volts.

When you input data to a TMS 9940 read/write memory using the TEST pin and P24 through P31. the address pointer is initialized to address 8300_{16}. The address keeps incrementing the high-to-low transition of each 50 millisecond programming pulse applied at pin 37. When you finally stop applying programming pulses. the last 16 bits of data input are interpreted as the beginning address for the program to be executed. This address may point to a read/write memory location. or to a read/write memory location. That is to say. the test program may be in read/write memory. in read-only memory. or in both areas.

THE TMS 9940 INSTRUCTION SET

The TMS 9940 instruction set is identical to the TMS 9900 instruction set, with these exceptions:

1) **The RSET, CKOF, CKON and LREX instructions have been deleted.** That is. all the external instructions except IDLE.

2) The XOP instructions will not work with operands 0, 1, 2, or 3.

3) There are new DCA and DCS instructions that enable 8-bit binary-coded decimal arithmetic.

Assuming that you start with two valid 8-bit binary-coded decimal operands, you can add these two 8-bit operands using normal binary addition. The result will be a meaningless 8-bit number; however, if you immediately execute the DCA instruction, this meaningless 8-bit number will be converted to a meaningful 8-bit, 2-BCD-digit number.

DCS, likewise, allows you to perform 8-bit binary-coded decimal subtraction. Assuming that the subtrahend and minuend are both valid 8-bit binary-coded decimal numbers, you perform a subtraction using binary arithmetic and you generate a meaningless 8-bit result. By executing the DCS instruction, you convert this meaningless 8-bit result into a valid 8-bit, 2-BCD-digit binary-coded decimal difference.

The DCA and DCS instructions both generate in the low-order eight bits of the 16-bit word.

For a discussion of decimal adjust logic in BCD addition or subtraction, see Volume 1, Chapter 3.

The LIIM instruction loads a 2-bit interrupt mask into the two low-order bits of the Status register.

Here are the instruction object codes used by the DCA, DCS, and LIIM instructions:

Instruction	Object Code	Bytes	Clock Periods
DCA r	0010110000bbssss	2	7
DCS r	0010110001bbssss	2	7
LIIM n	001011001xxxxxnn	2	10

The object code notation above conforms to that which we have described for Table 18-3. For the LIIM instruction, x represents "don't care" bits and n represents the two binary digits that get loaded into the two low-order Status register bits.

THE TIM 9904 FOUR-PHASE CLOCK GENERATOR/DRIVER

This part is also given the generic TTL name: the SN74LS362. The TIM 9904 provides TMS 9900 microprocessors with the four clock signals: Φ1, Φ2, Φ3, and Φ4. These are +12V MOS driver signals. In addition, four complementary +5V clock signals, $\overline{\Phi 1}$, $\overline{\Phi 2}$, $\overline{\Phi 3}$, and $\overline{\Phi 4}$, are generated for use elsewhere in a TMS 9900 microcomputer system.

The TIM 9904 device may be driven by an external crystal, an external LC circuit, or a single external clock signal.

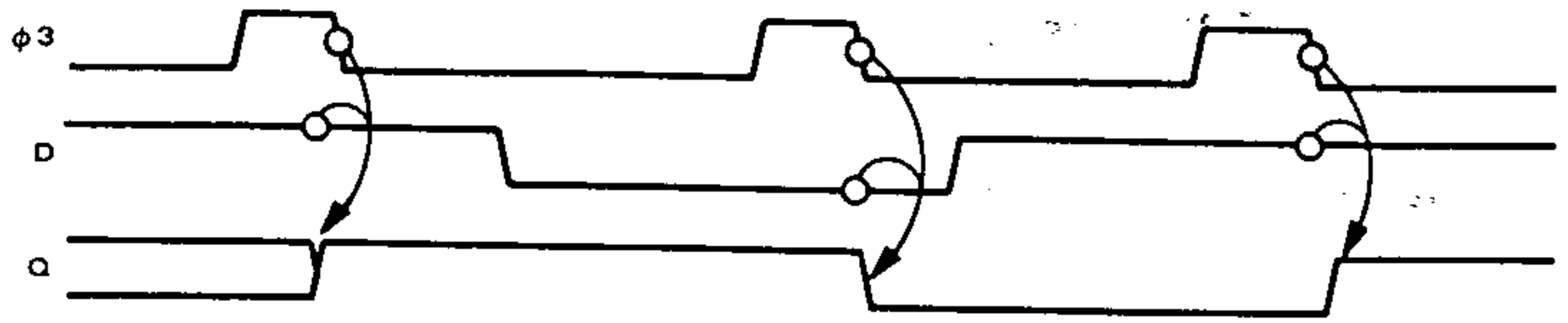
The TIM 9904 is manufactured using low-power Schottky technology; hence the 74LS part number. It is packaged as a 20-pin DIP. All signals, other than the four MOS level clocks, are TTL-compatible.

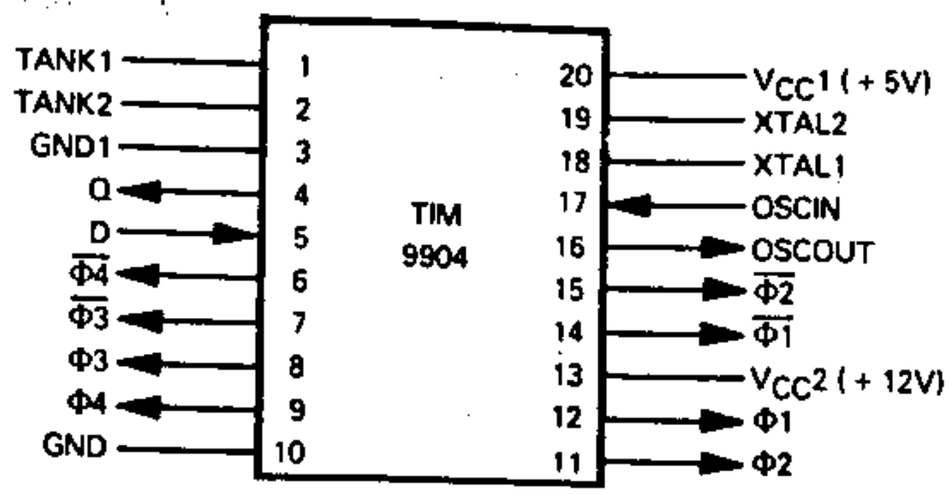
The TIM 9904 allows one asynchronous input signal to be synchronized, via a D flip-flop, with the Φ3 signal. The synchronized signal is output, frequently to be used as a $\overline{\text{RESET}}$ input to the TMS 9900.

Figure 18-23 illustrates TIM 9904 pins and signal assignments.

The four clock signals, Φ1,Φ2, Φ3, and Φ4, conform to Figure 18-3. Φ1, Φ2, Φ3, and Φ4 are complements of $\overline{\Phi 1}$, $\overline{\Phi 2}$, $\overline{\Phi 3}$, and $\overline{\Phi 4}$.

A logic level input at D will be output at Q on the high-to-low transition of Φ3:

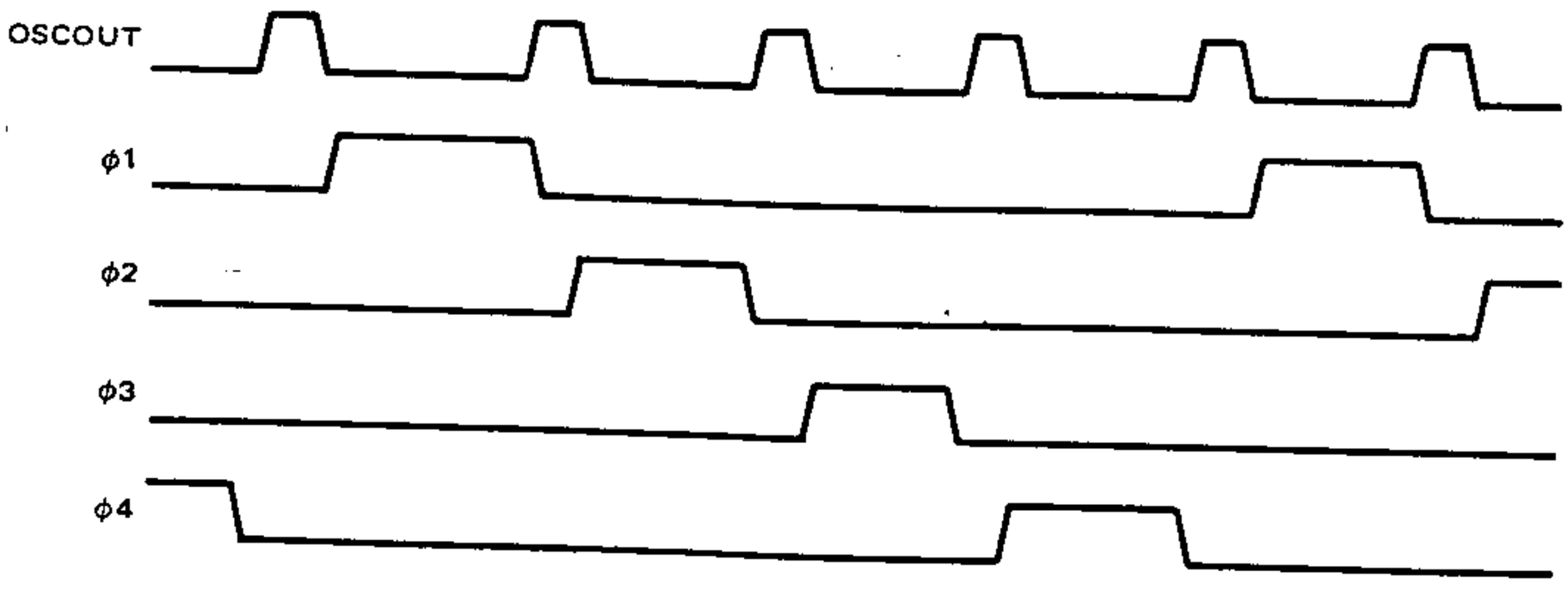




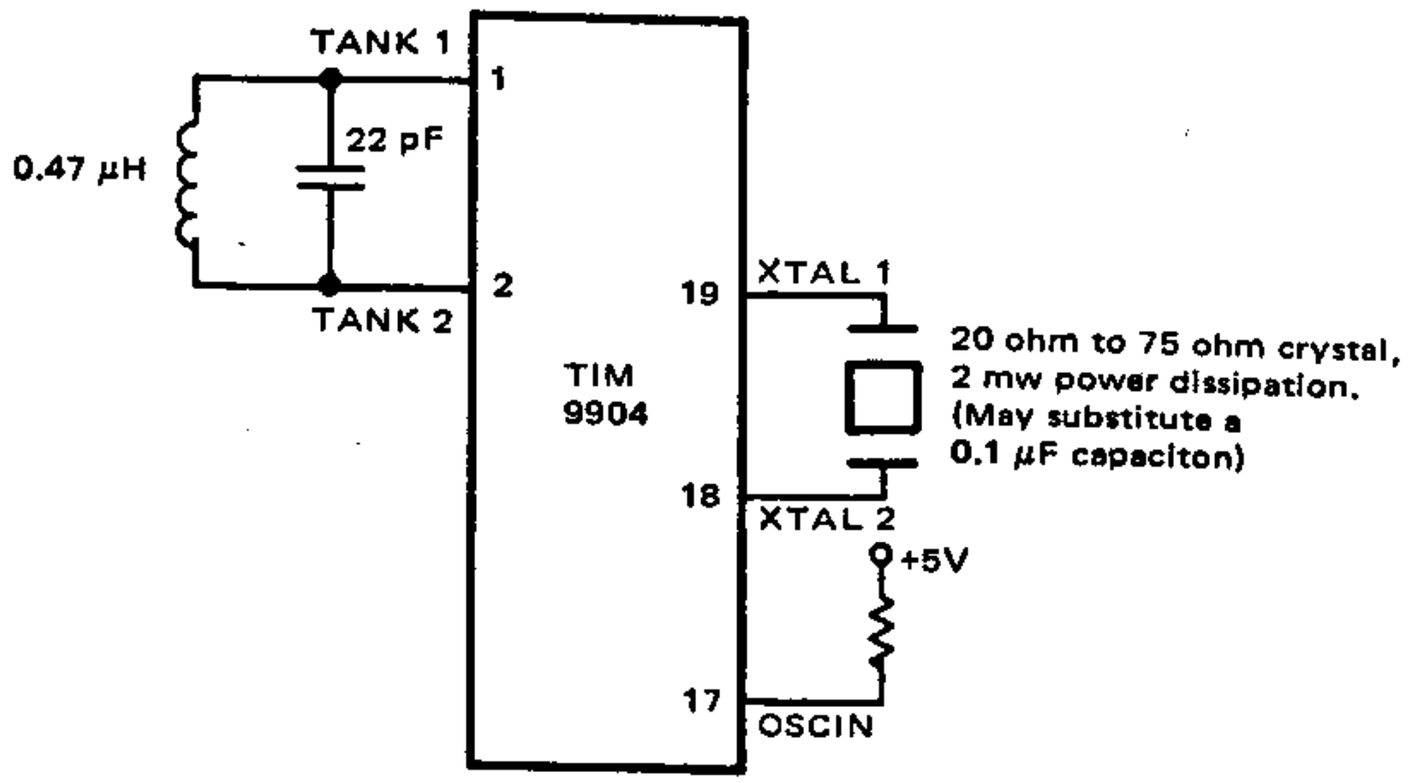
Pin Name	Description	Type
Φ1, Φ2, Φ3, Φ4	+ 12V clocks to drive a TMS 9900	Output
Φ1, Φ2, Φ3, Φ4	+ 5V clock complements	Output
D	Asynchronous control	Input
Q	Synchronized control	Output
TANK1, TANK2	Crystal overtone controls	
XTAL1, XTAL2	External crystal connections	
OSCIN	External clock	Input
OSCOUT	Clock with frequency 4Φ	Output
$V_{CC}1$, $V_{CC}2$, GND1, GND2	Power, Ground	

Figure 18-23. TIM 9904 Signals and Pin Assignments.

OSCOUT provides a clock frequency four times that of the Φ clocks. Its phase relationship to the Φ clocks may be illustrated as follows:



When an external quartz crystal is used to drive the TIM 9904, the following connections are required:



OSCIN must be tied to a high logic level for the internal clock logic to work properly.

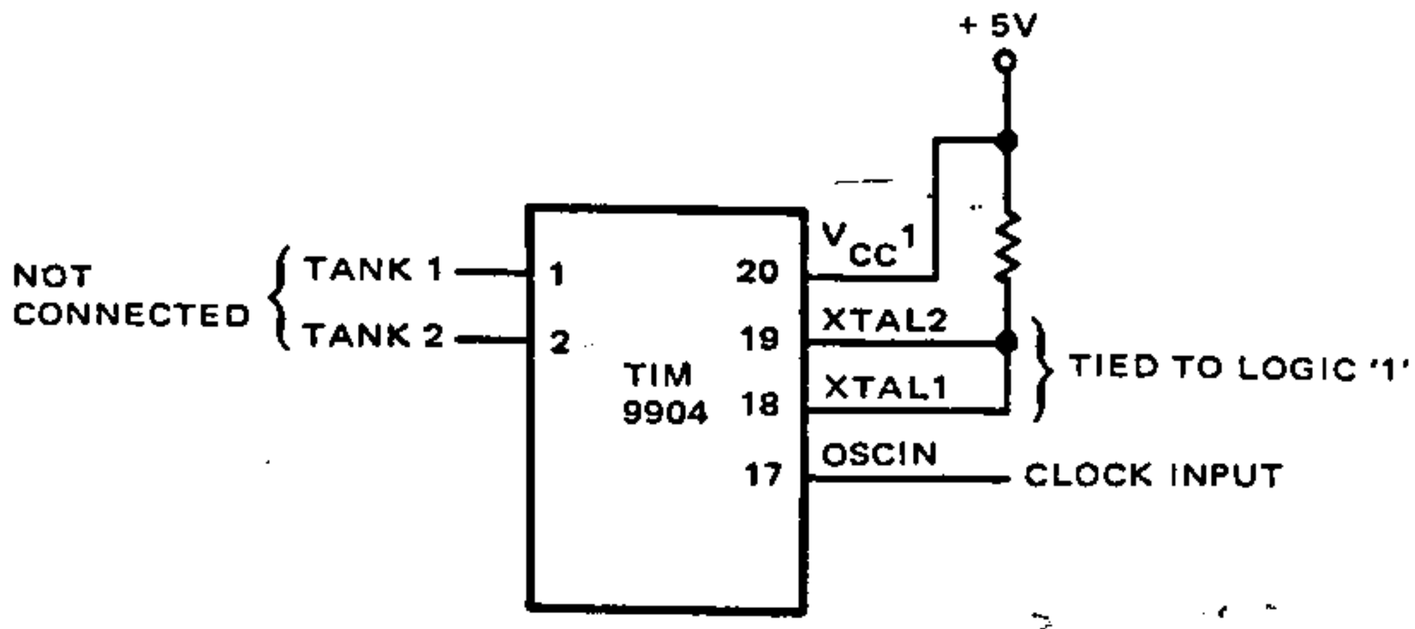
Required capacitor and inductance values are shown in the illustration above for a TMS 9900 microprocessor operating with its standard 3 MHz frequency. The crystal must have a resonant frequency of 48 MHz. For 48 MHz operation, a third overtone crystal is used.

For less precise timing, the quartz crystal may be replaced with a 0.1 μf capacitor. The LC-tuned circuit now establishes the clock frequency according to the following equation:

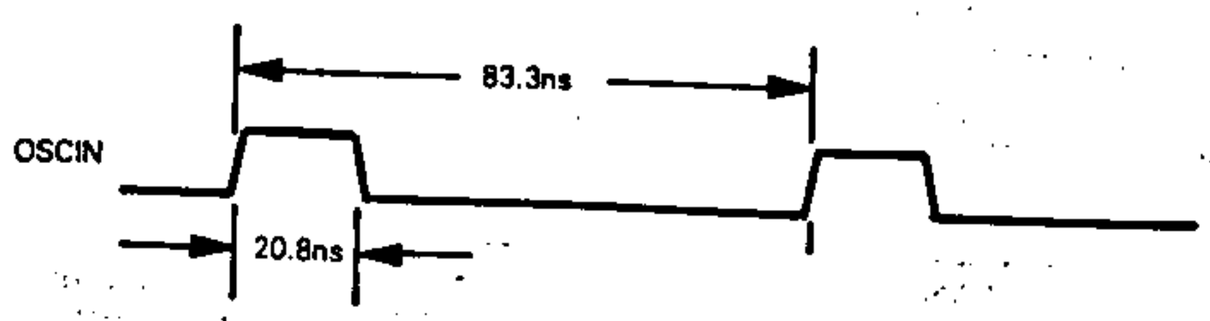
$$f_{osc} = 1/(2\pi\sqrt{LC})$$

where L is the inductance, with units of Henries, and C is the capacitance with units of Farads. This includes the capacitance of the circuit into which the components are mounted.

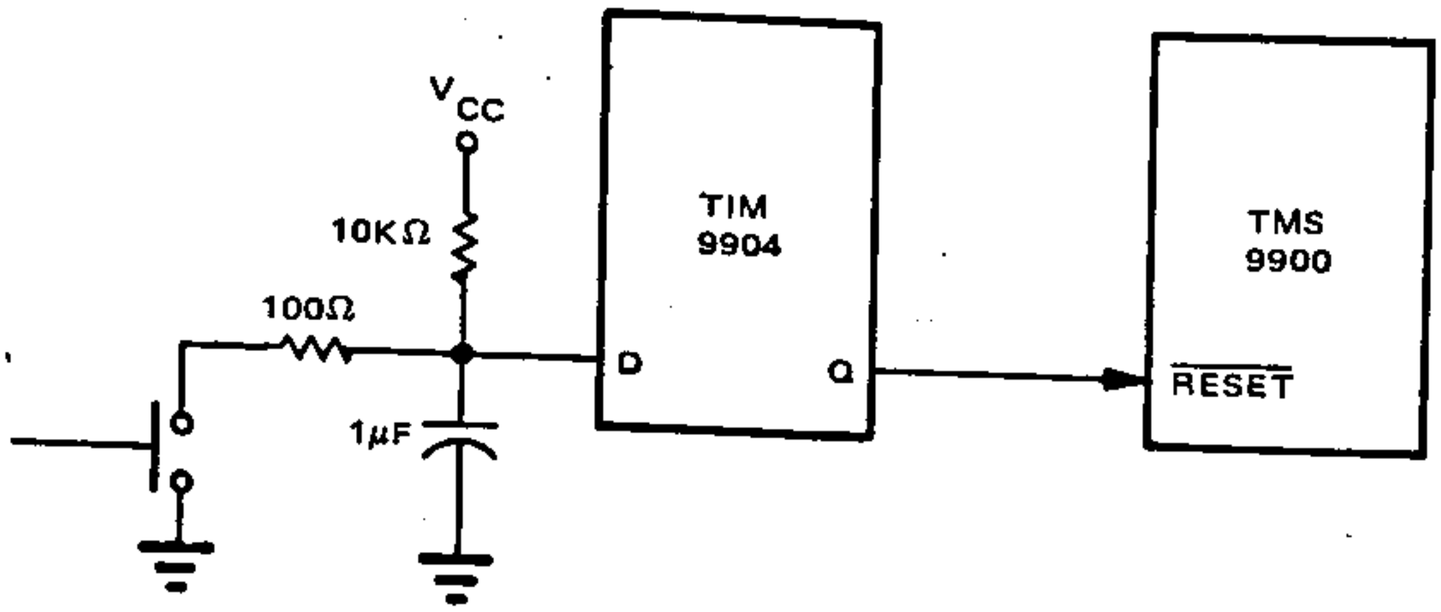
If an external clock signal is input, it must occur at OSCIN. The crystal connections XTAL1 and XTAL2 should be connected to V_{CC} as follows:



The clock input OSCIN must have a frequency which is four times the clock period frequency and has a 25% duty cycle. Thus. for a 3 MHz frequency. a 12 MHz signal must be input via OSCIN:



In TMS 9900 microcomputer systems, the D input is used for an asynchronous reset; Q is output as a synchronous reset. This may be illustrated as follows:



The illustration above shows recommended resistor and capacitor values.

THE TMS 9901 PROGRAMMABLE SYSTEM INTERFACE (PSI)

The TMS 9901 Programmable System Interface (PSI) is a special support part designed for the TMS 9900 series of microprocessors. This relatively primitive device uses 32 bits of the TMS 9900 CRU bit field to support parallel I/O and interrupt request logic. Programmable timer logic is also available.

Figure 18-24 illustrates that part of general microcomputer system logic which has been implemented on the TMS 9901 PSI.

The TMS 9901 PSI is packaged as a 40-pin DIP. It uses a single +5V power supply. All inputs and outputs are TTL-compatible. The device is implemented using N-channel silicon gate MOS technology.

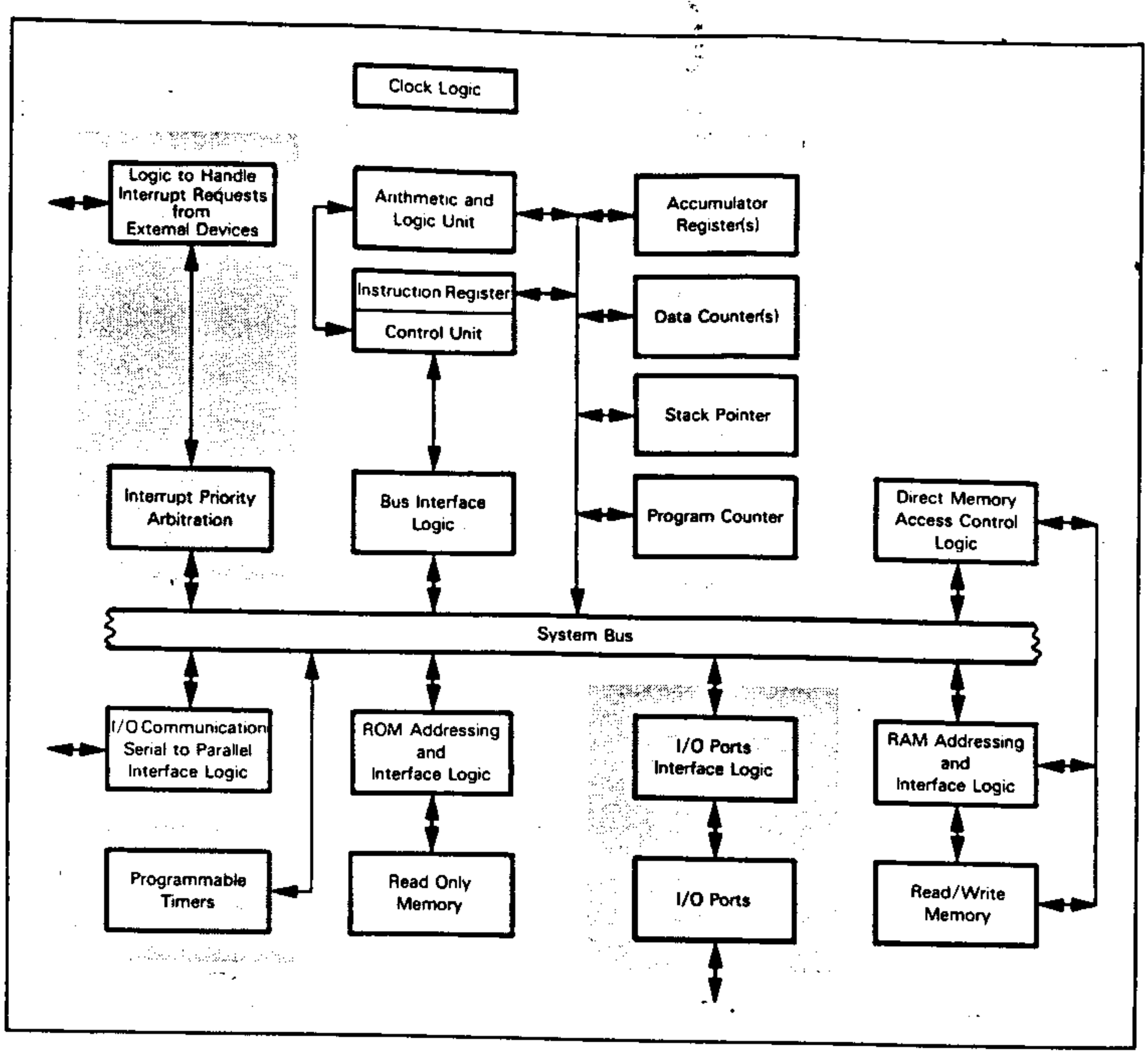
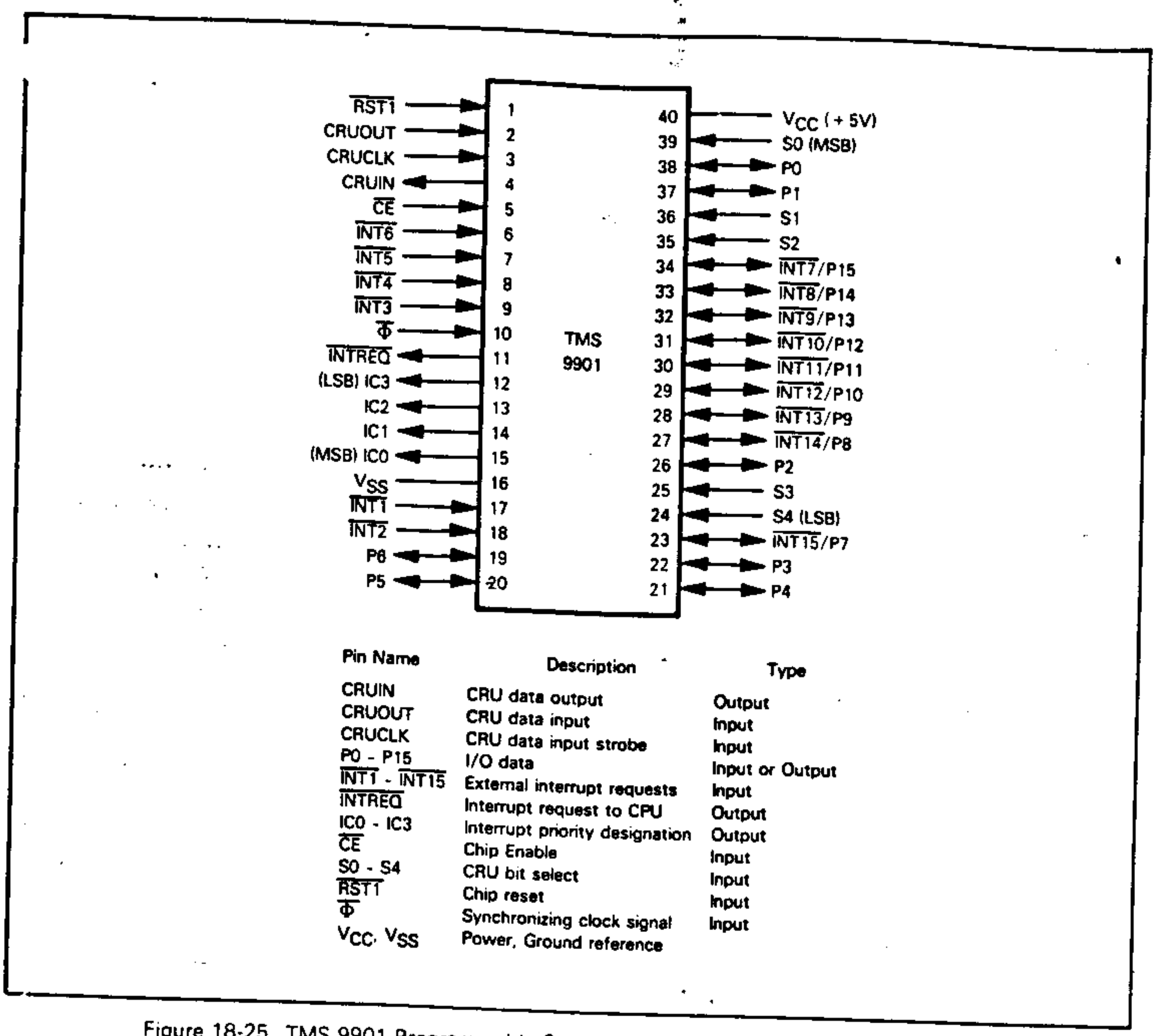


Figure 18-24. Logic of the TMS 9901 Programmable System Interface



Pin Name	Description	Type
CRUIN	CRU data output	Output
CRUOUT	CRU data input	Input
CRUCLK	CRU data input strobe	Input
P0 - P15	I/O data	Input or Output
INT1 - INT15	External interrupt requests	Input
INTREQ	Interrupt request to CPU	Output
IC0 - IC3	Interrupt priority designation	Output
CE	Chip Enable	Input
S0 - S4	CRU bit select	Input
RST1	Chip reset	Input
Φ	Synchronizing clock signal	Input
VCC, VSS	Power, Ground reference	

Figure 18-25. TMS 9901 Programmable System Interface Signals and Pin Assignments

In the illustration above, Address lines have been numbered using our standard notation, whereby A14 is the highest-order address line and A0 is the lowest-order address line. This is the opposite of Texas Instruments' notation. The CRU select lines are numbered according to Texas Instruments' notation and Figure 18-25. Therefore, S4 is connected to A0, and S0 is connected to A4.

TMS 9901 PSI PINS AND SIGNALS

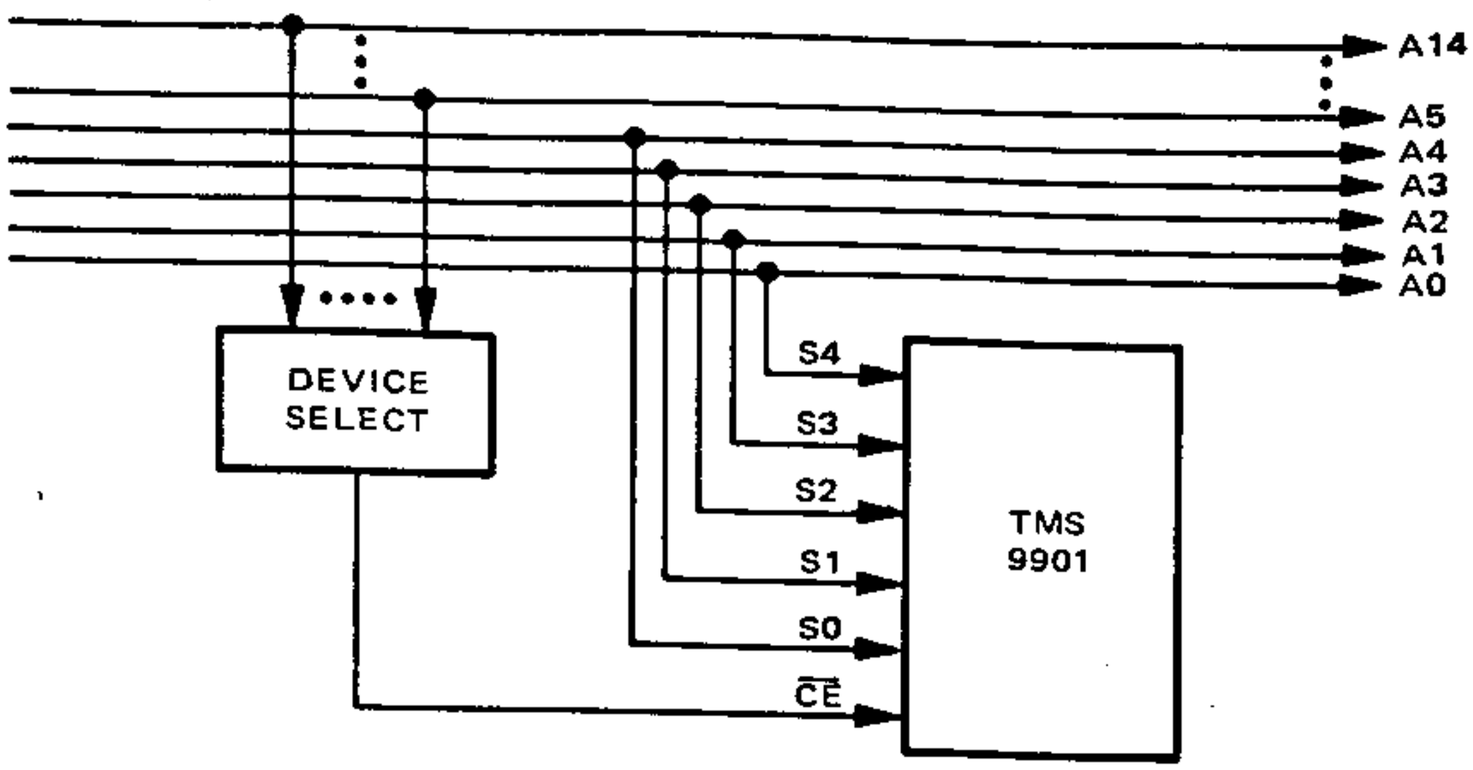
The TMS 9901 pins and signals are illustrated in Figure 18-25. The signals which connect the TMS 9901 to a TMS 9900 series microprocessor are quite straightforward: they **consist of the CRU and interrupt signals.**

The CRU signals include CRUIN, CRUOUT, and CRUCLK.

The interrupt signals consist of $\overline{\text{INTREQ}}$, IC0, IC1, IC2, and IC3.

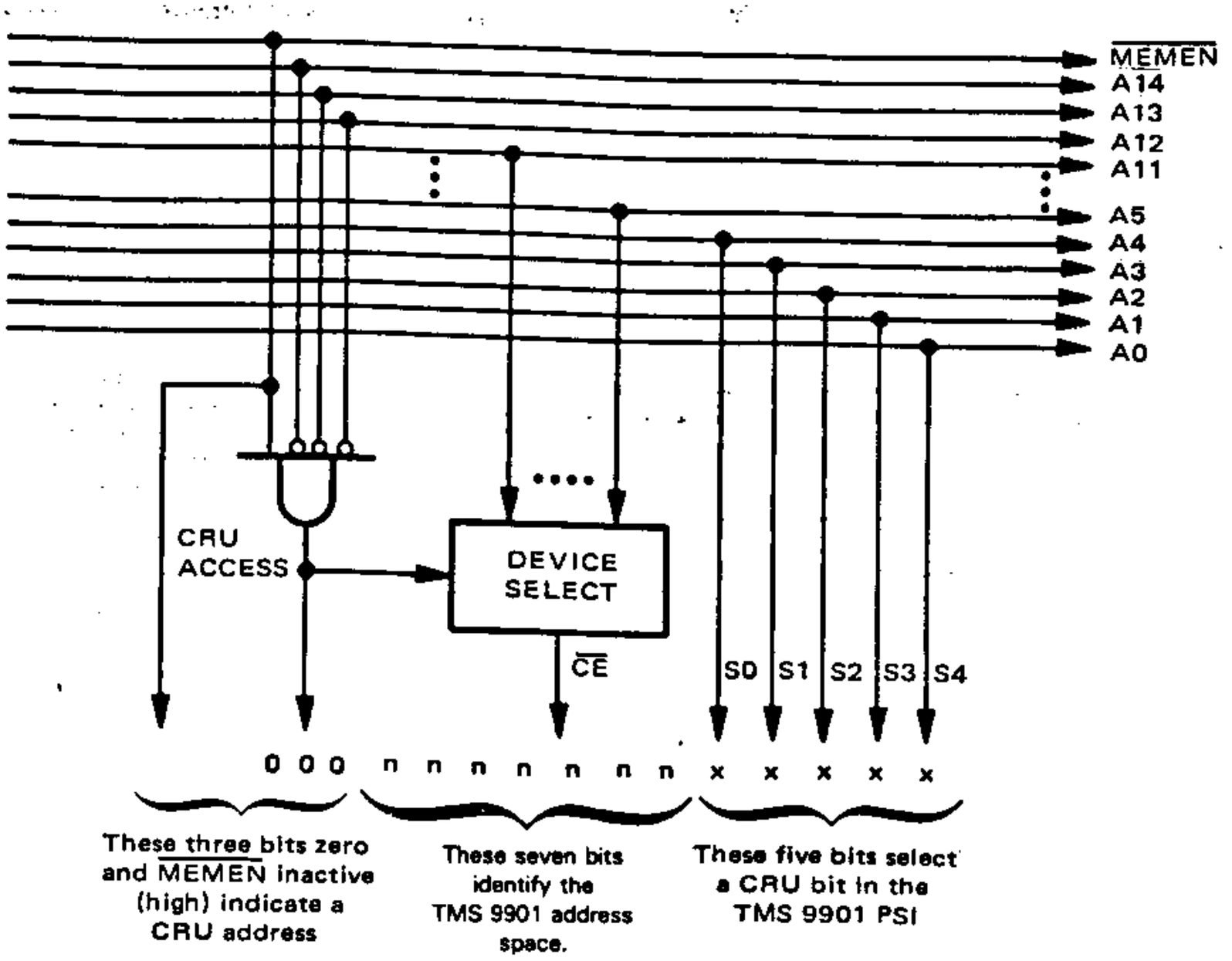
For a description of CRU and interrupt signals, refer back to our TMS 9900 discussion.

Device select logic includes a chip enable input, $\overline{\text{CE}}$, together with five CRU bit select pins, S0 - S4. $\overline{\text{CE}}$ and S0 - S4 will connect to the Address Bus as follows:



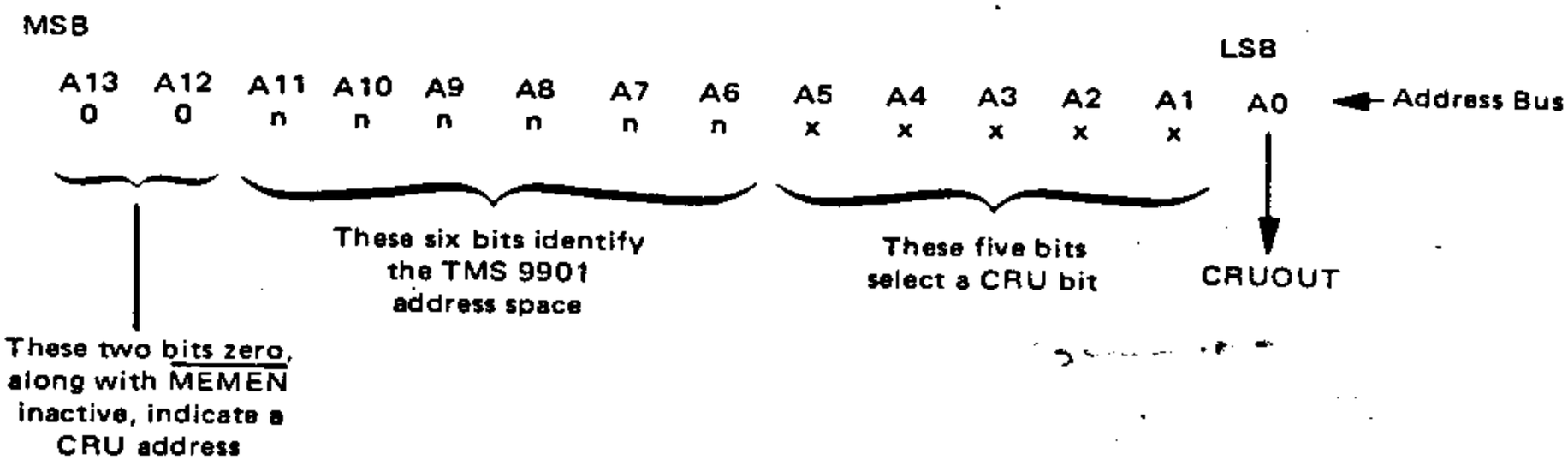
In the illustration above, Address lines have been numbered using our standard notation, whereby A14 is the highest-order address line and A0 is the lowest-order address line. This is the opposite of Texas Instruments' notation. The CRU select lines are numbered according to Texas Instruments' notation and Figure 18-25. Therefore, S4 is connected to A0, and S0 is connected to A4.

Device select logic determines the CRU address space that will be reserved for the TMS 9901 PSI. This may be illustrated as follows:



The high-order three address lines, which we call A14, A13, and A12, are all zero during a CRU access, at which time MEMEN is inactive (high). Thus we decode address lines A11 through A5 to select a particular TMS 9901 device.

Since the TMS 9980 uses the Address Bus differently during a CRU operation, TMS 9901 device select logic would connect to the Address Bus in a different way. The CRU bit select lines S0 - S4 would be tied to lines A5 - A1; device select logic would decode lines A11 - A6; and lines A13 and A12, along with MEMEN, would indicate a CRU access. We illustrate this as follows:



$\overline{\Phi}$ is a synchronizing clock signal used to time data output and to sample interrupts. $\overline{\Phi}$ is the complement of $\Phi 3$. For the TMS 9900, $\overline{\Phi 3}$ is generated by the TMS 9904. The TMS 9980 outputs $\overline{\Phi 3}$ directly.

The best way of understanding the interface between a TMS 9901 and external logic is to look at functions performed, as illustrated in Figure 18-26.

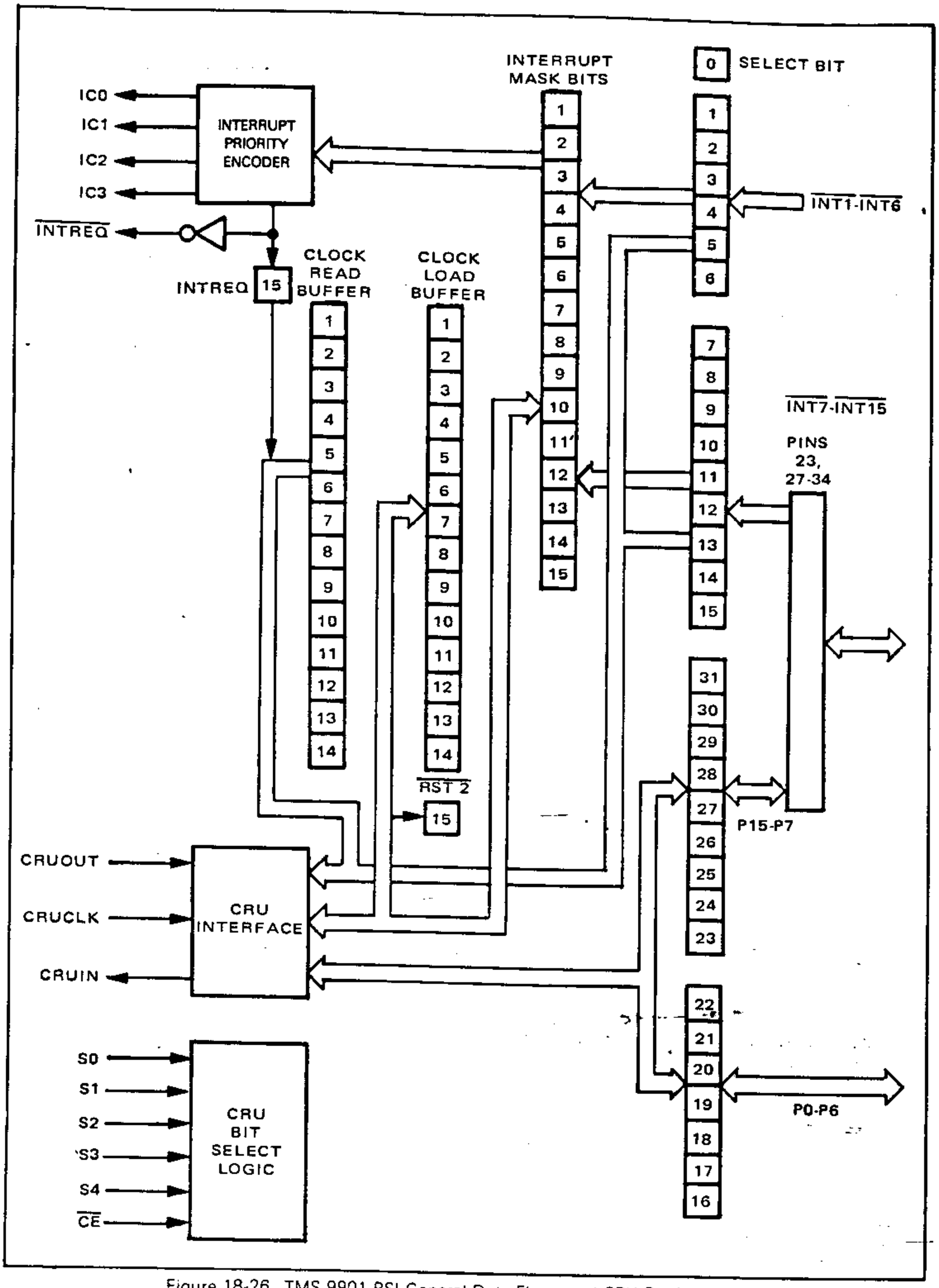


Figure 18-26. TMS 9901 PSI General Data Flows and CRU Bit Assignments

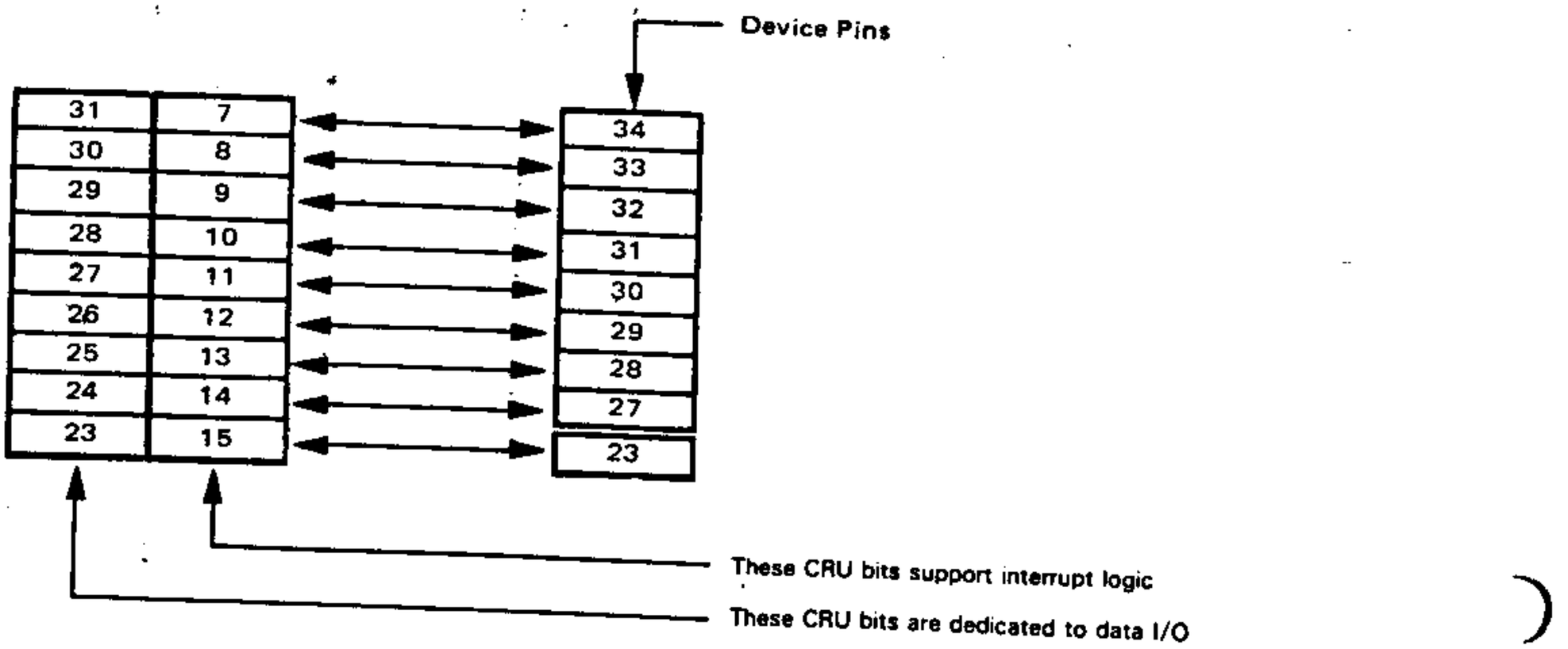
From the programmer's viewpoint, a TMS 9901 looks like 32 contiguous CRU bits. Thus, you will access any part of a TMS 9901 device's logic using CRU input and output instructions.

As you read through the TMS 9901 description that follows, you should bear in mind the power of multi-bit CRU load and store instructions as they apply to TMS 9901 architecture. A single instruction transferring an appropriate bit pattern can frequently perform multiple control and data transfer operations.

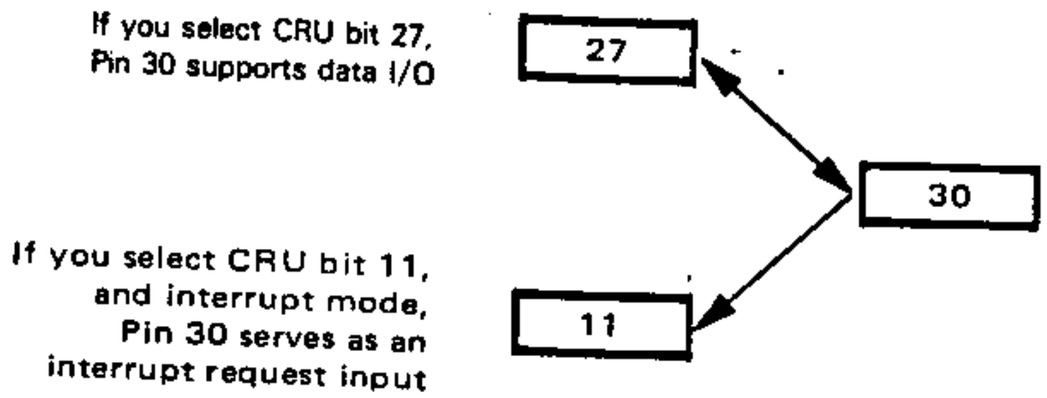
The manner in which CRU bits are used by the TMS 9901 is not straightforward. This is because CRU bits share functions and pins. Functions and pins are shared in different ways.

Let us first look at pin connections. CRU bits 1-6 connect to pins $\overline{INT1}$ - $\overline{INT6}$; thus, in interrupt mode each of these CRU bits has its own dedicated input pin.

CRU bits 7-15 share nine input or output pins with CRU bits 23-31. CRU bits share pins as follows:



Each of the CRU bits shown above shares a pin with another CRU bit. That is to say, within the illustrated CRU address range, there are two CRU bits which will access the same pin, although each CRU bit performs a different operation. Thus you use the same pin in one of two different ways, using a bit address to select one operation. This may be illustrated as follows:



CRU bits 16-22 connect to parallel I/O pins. These bit addresses are not shared with any other TMS 9901 functions.

CRU bit 0 is a select bit that is not connected to any pin. A 1 written into this bit causes bits 1-15 to support real-time clock logic. A 0 written into CRU bit 0 selects interrupt logic. When CRU clock logic is selected, bits 1-14 function as two 14-bit real-time Clock Buffer registers — one a read-only register, the other write-only. Real-time clock logic is separate from, and operates simultaneously with, and/or parallel I/O logic. That is to say, the process of selecting real-time clock logic does not disable any other logic. The select bit merely chooses which registers CRU addresses will access, rather than enabling or disabling any operations.

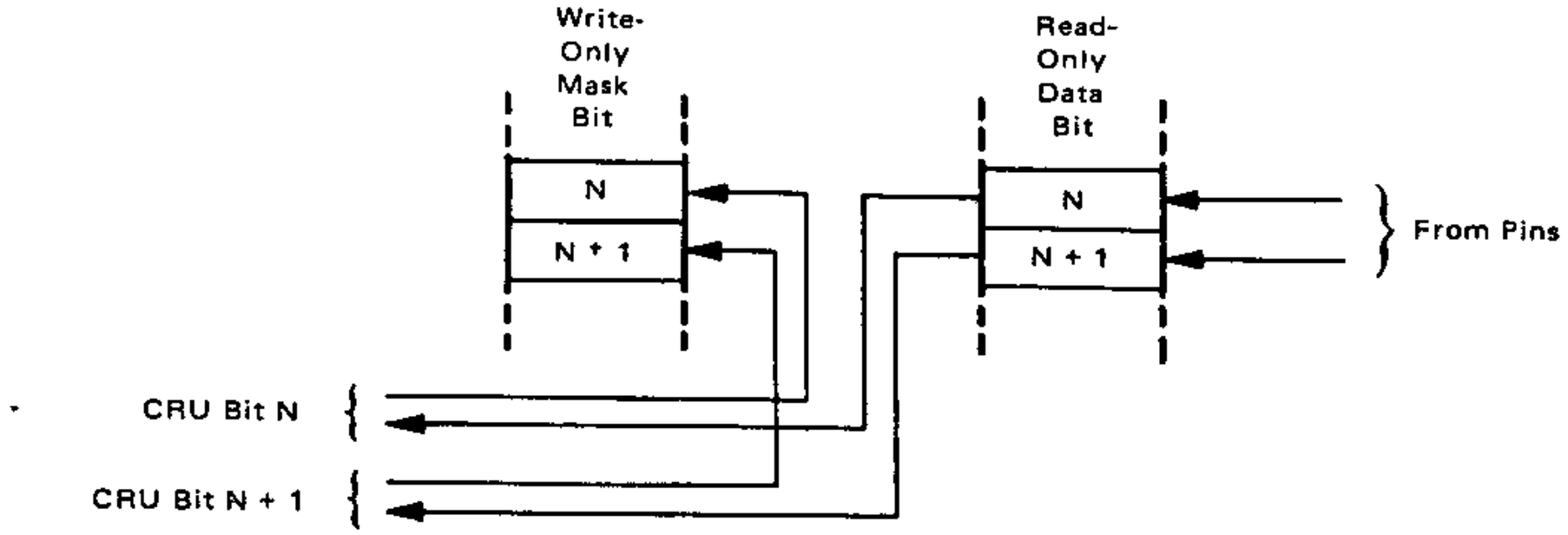
TMS 9901 PSI INTERRUPT LOGIC

The easiest place to start understanding the TMS 9901 is at its interrupt logic.

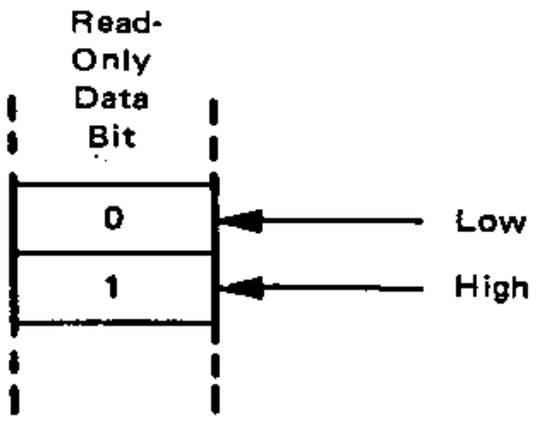
External logic can input data to CRU bits 1-15 via their connected pins. These input data signals will be interpreted as interrupt requests if interrupts are enabled. If interrupts are disabled, then these CRU bits act simply as data input.

You access interrupt logic through the CRU when the select bit, CRU bit 0, contains a 0.

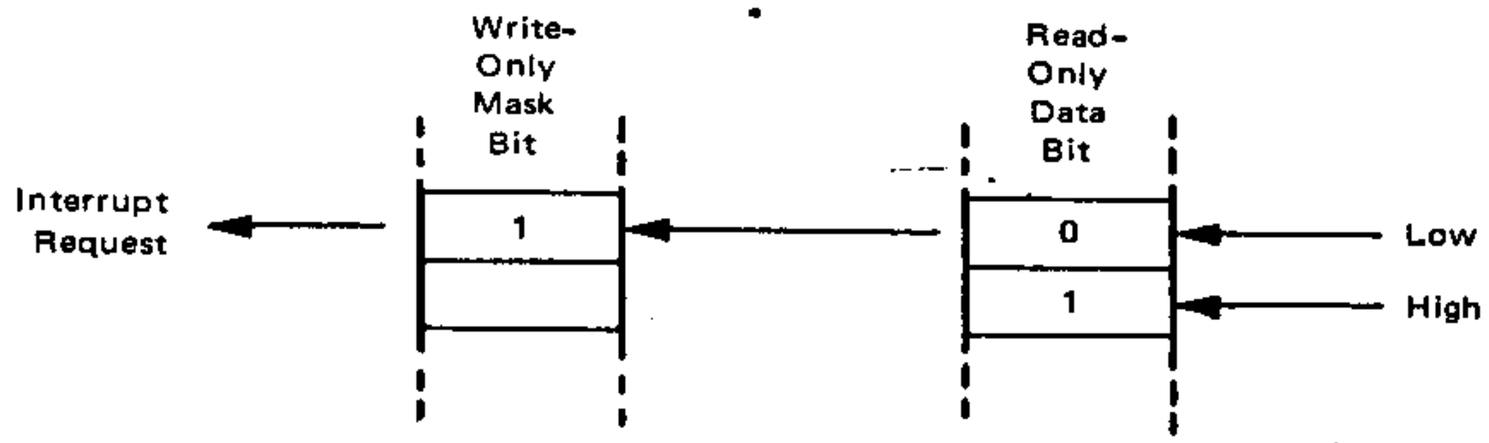
CRU bit addresses 1-15 each access separate read-only and write-only locations. The read-only location stores the signal level input at the attached pin. The write-only location accesses an interrupt mask bit. This may be illustrated as follows:



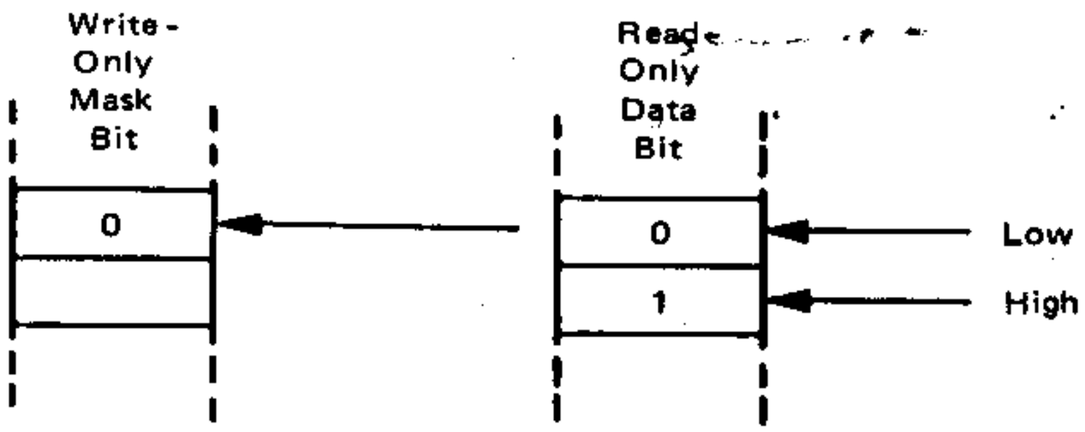
Signals arriving at pins connected to CRU bits 1-15 are immediately reflected by CRU bit contents:



A low level (that is, a 0 bit) is interpreted as an interrupt request. The interrupt request is passed on to the mask bit. If the mask bit contains 1, the interrupt is enabled and the interrupt request is passed on:



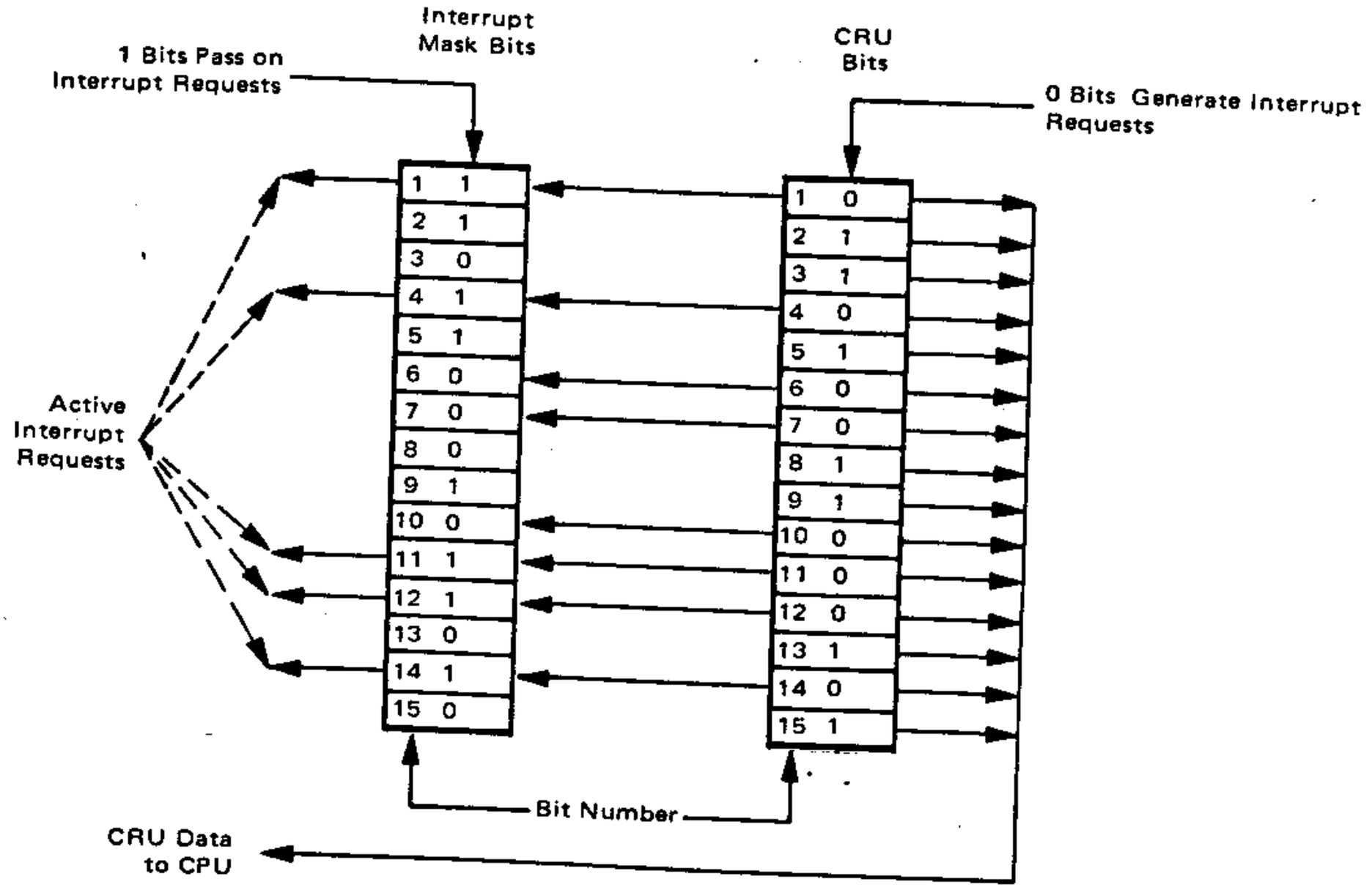
If the mask bit is 0, the interrupt request is disabled and therefore denied:



Quite apart from interrupt logic, the CPU can at any time read the contents of one or more CRU bits in the address range 1-15. Here are some instructions that may access CRU bits 1-15 in various ways:

```
        LI      R12,PSI+1   LOAD CRU BASE ADDRESS INTO R12
        LI      R1,MASK     LOAD INTERRUPT MASK BITS INTO R1
        LDCR    R1,15       OUTPUT TO WRITE-ONLY MASK LOCATIONS
        -
        -
        -
        STCR    R2,15       INPUT CRU BITS 1 THROUGH 15 AS DATA TO R2
        -
        -
        -
```

For some randomly selected data levels, CRU bits 1-15 may be illustrated as follows:



If one or more CRU bit's interrupt requests are low, and the corresponding mask bit is 1, then interrupt priority encoder logic outputs $\overline{\text{INTREQ}}$ low. Simultaneously, the level of the active interrupt request which has highest priority is identified via IC0 - IC3.

$\overline{\text{INT1}}$, input to CRU bit 1, has highest priority;
$\overline{\text{INT15}}$, input to CRU bit 15, has lowest priority.

The levels at IC0 - IC3 are maintained until the interrupt request signal is removed at the external pin, or the interrupt mask bit for the level is reset to 0.

TMS 9901 PSI DATA INPUT AND OUTPUT

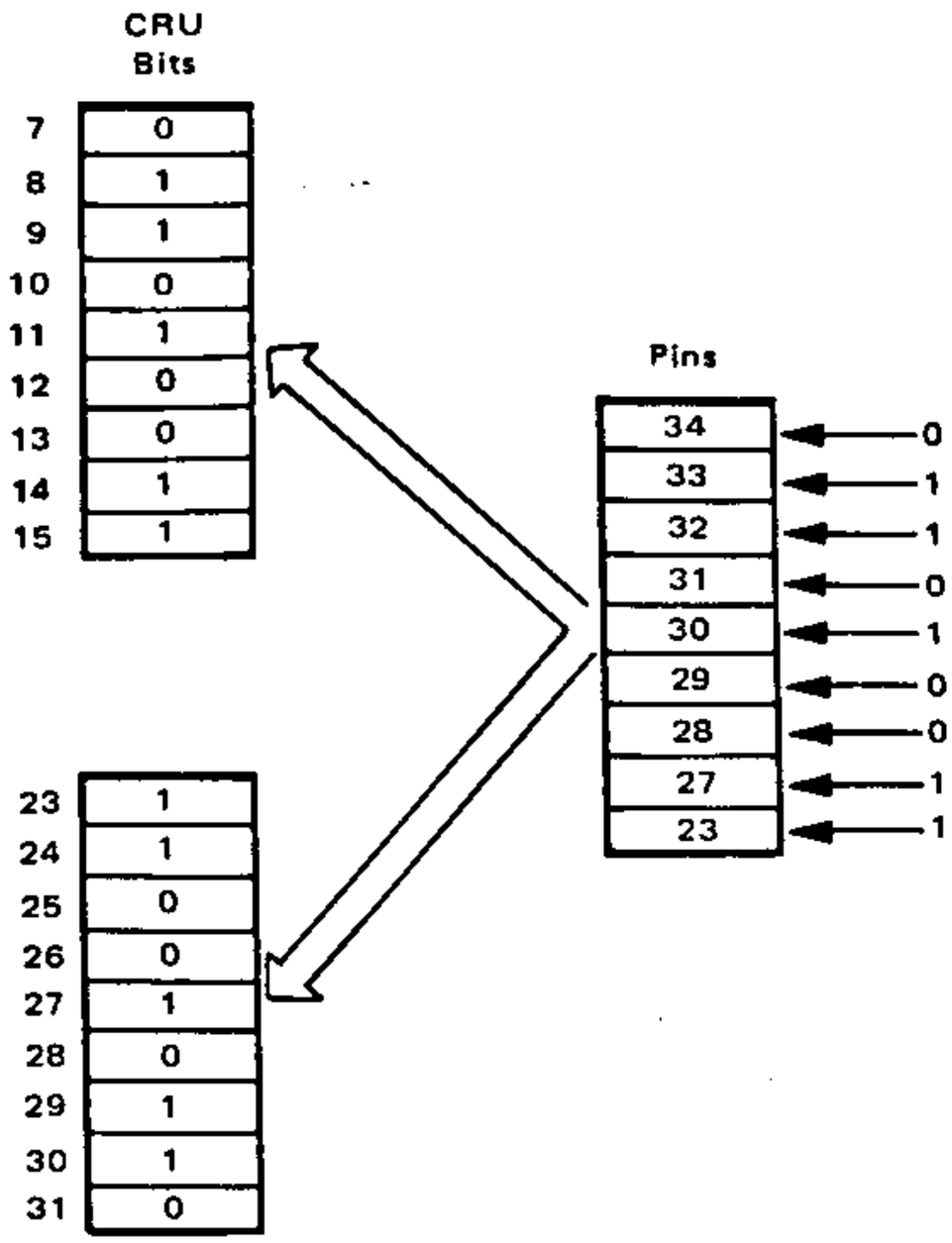
You can use CRU I/O instructions to input, output, or test external data at CRU bits 16-31. Data is output from the CPU to the TMS 9901 via CRUOUT; it is input from the TMS 9901 to the CPU via CRUIN. Bits are addressed via S0 - S4, as we have already described.

Following a reset, pins connected to CRU bits 16-31 are in input mode. In this mode, external logic can assert high or low levels at connected pins, in which case one or two CRU bits will be affected: a signal input to P0 - P6 will generate data in CRU bits 16-22; if interrupt mode is selected (by a 0 in CRU bit 0), a signal input to $\overline{\text{INT7}}$/P15-$\overline{\text{INT15}}$/P7 will

generate data in two CRU bits, one in the CRU bit range 7-15, the other in CRU bit range 31-23. **In interrupt mode, if the CPU inputs data from CRU bits 7-15 or 31-23, then it will input the same data, but in reverse order.** This may be illustrated as follows:

```
LI    R12,PSI+7
STCR  R1,9
```

0196_{16} Loaded into R1




```
LI    R12,PSI+23
STCR  R1,9
```

$00D3_{16}$ Loaded into R1

Note that, as in all CRU transfers, the first CRU bit transferred goes to the least significant bit position of the destination register.

As soon as the CPU outputs data to any bit capable of supporting data output, the I/O logic associated with this bit is put into output mode. In this mode, a pin will output a voltage level reflecting data in the corresponding CRU bit. External logic cannot input data to a CRU bit that is in output mode; in fact, driving input currents into an output pin may damage the TMS 9901.

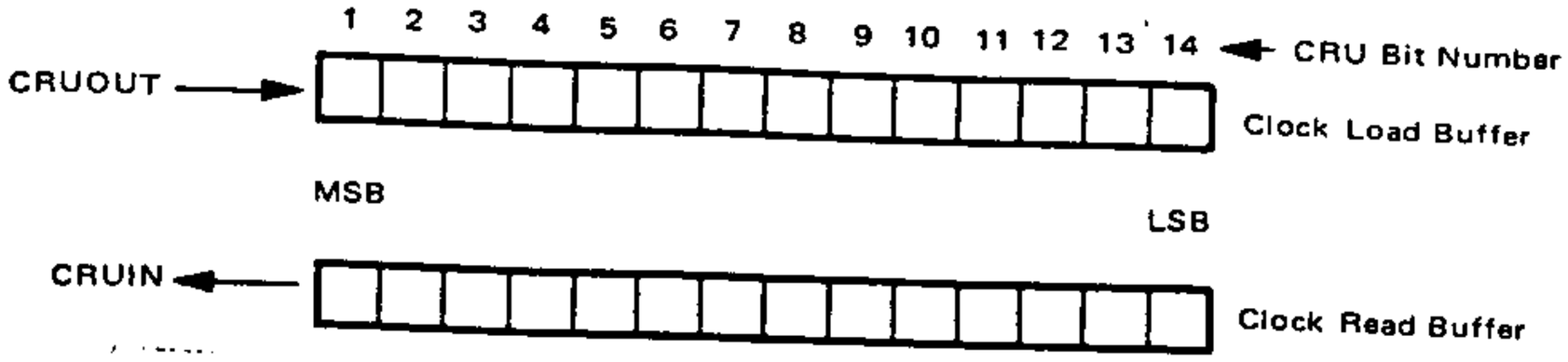
Once a CRU bit has been placed in output mode, it remains in output mode until the TMS 9901 is reset. That is to say, you cannot selectively return CRU bits from output mode to input mode. However, you can always read output bits back to the CPU; that is, **although external logic must never attempt to input to a pin that is in output mode, the CPU can always read the contents of any I/O bit, whether it is an input or an output.**

You cannot output data via CRU bits 7-15, even though these bits are connected to the same pins as CRU bits 31-23. When you output data to CRU bits 7-15, the data is routed to one of two write-only locations, depending on the contents of CRU bit 0: if the select bit is 0, the data goes to interrupt mask bits $\overline{7\text{-}15}$; if clock mode is selected (CRU bit 0 contains 1), the data goes to the Clock Load Buffer register (bits 7-14) and $\overline{RST2}$ (bit 15).

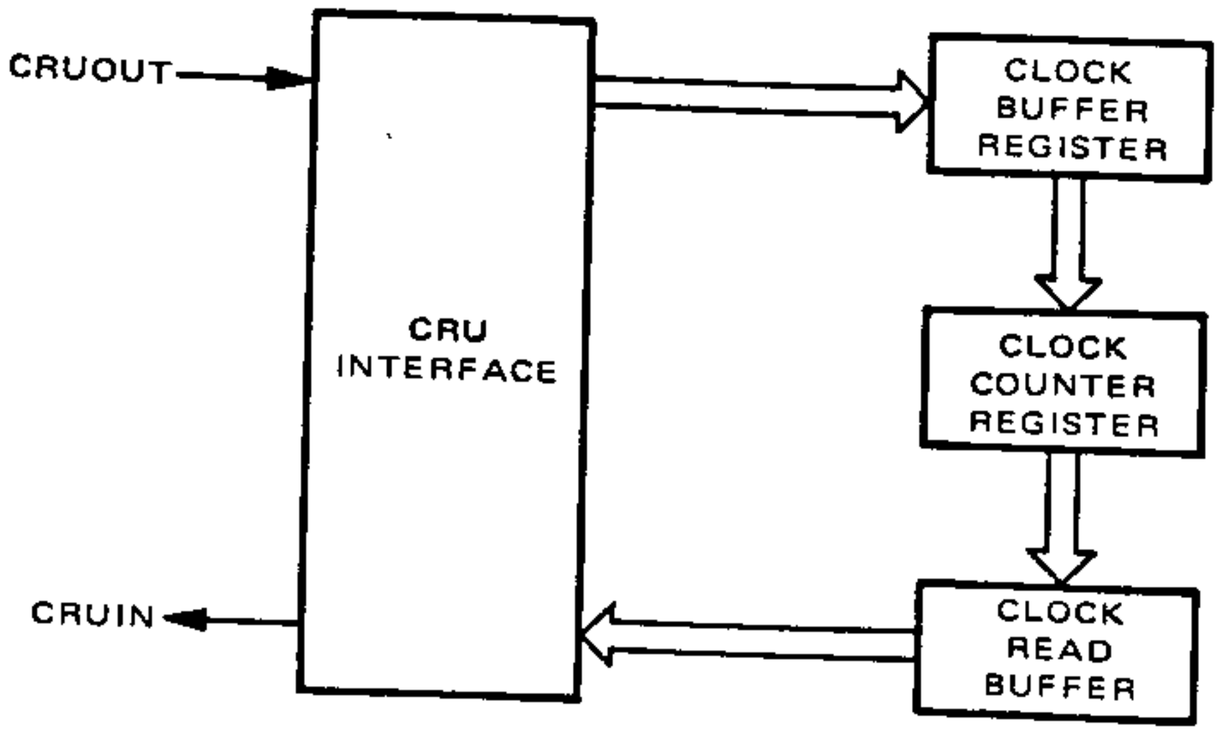
In interrupt mode you can input external data from CRU bits 1-6. Once again, you cannot output data via these CRU bit addresses, since any data output will be routed to corresponding interrupt mask bits or Clock Load Buffer bits.

TMS 9901 REAL-TIME CLOCK LOGIC

If you write a 1 into CRU bit 0 of a TMS 9901 device, then CRU bits 1-14 are used as two 14-bit Clock buffers, which may be illustrated as follows:



Besides these two buffers, **real-time Clock logic contains a decrementing register which we call the Clock Counter register.** The CPU loads the Clock Counter register via the Clock Load Buffer, and reads the Counter contents via the Clock Read Buffer. We illustrate this in the following way:



The Clock Counter register decrements continuously as long as the TMS 9901 is powered up. This will cause no problems as long as the clock interrupt is disabled.

When you write any non-zero value into the Clock Load Buffer (CRU bits 1-14), the Clock Counter register starts decrementing from that value. A decrement occurs once every 64 $\overline{\Phi}$ clock pulses. Thus, with a 3 MHz clock, a decrement occurs once every 21.3 microseconds. **When the CRU Clock Counter register decrements to 0, an interrupt request is generated, the previously output starting value is reloaded, and the clock starts to decrement again.** Thus, with a 21.3-microsecond time interval between decrements, the maximum time interval between interrupt requests will be 249 milliseconds.

An enabled clock interrupt request causes $\overline{\text{INTREQ}}$ to be output low, together with a level 3 interrupt identified via IC0 - IC3. That is to say, the $\overline{\text{INT3}}$ external interrupt and the Clock logic share the same interrupt level and interrupt mask bit. In clock mode, CRU bit 15 is used to record the state of the $\overline{\text{INTREQ}}$ signal. Thus, if interrupt requests are disabled, the CPU program can check for a time-out by testing the level at CRU bit 15. This bit will be low if no time-out has occurred, and it will be high if a time-out has occurred; thus this bit is the complement of $\overline{\text{INTREQ}}$.

Following a CRU real-time clock interrupt request, you must write into interrupt mask bit 3 in order to clear the interrupt request. You can write a 0 or a 1 into the interrupt mask bit. Normally, you will write a 1 in order to keep interrupts enabled. Writing a 0 will clear any active real-time clock interrupt request, and will simultaneously disable further real-time clock interrupt requests.

The Clock Read Buffer register contents do not change as long as the TMS 9901 is in clock mode. This characteristic insures that the Clock Read Buffer will hold a stable value while the CPU is reading it — even though the Clock Counter may decrement during the read operation.

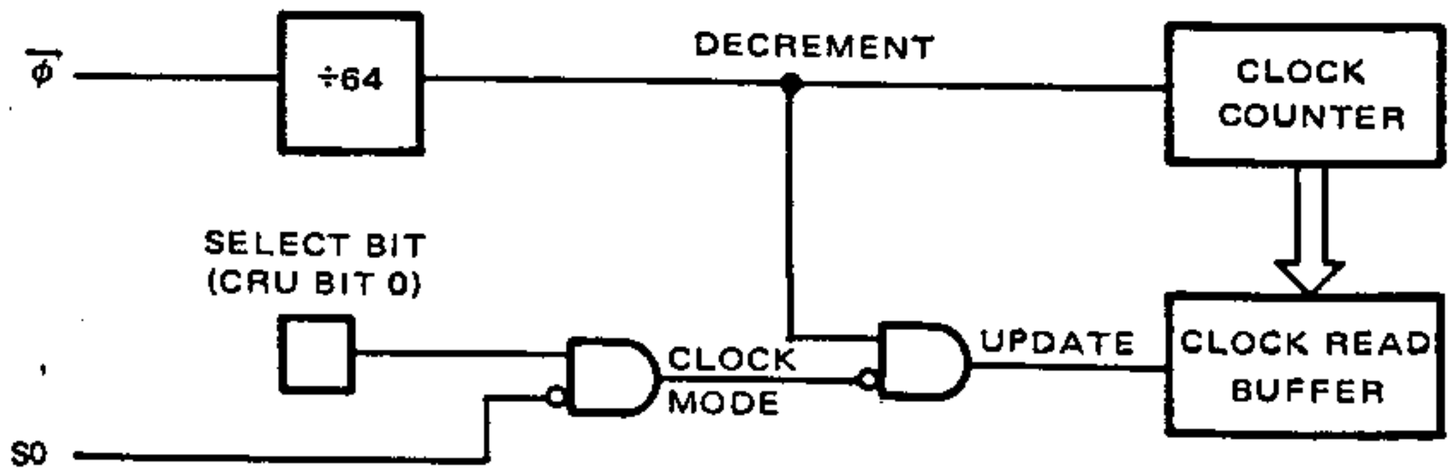
Either of the following two events will cause the Clock Counter contents to transfer to the Clock Read Buffer:

- The $\overline{\Phi}$ pulse which causes the Clock Counter to decrement.
- An exit from clock mode.

Thus, the Clock Read Buffer register is updated whenever the TMS 9901 leaves clock mode, and every time the Clock Counter decrements outside of clock mode.

Beware — even if CRU bit 0 contains a 1, the TMS 9901 will exit clock mode for as long as it sees a 1 on select line S0; this will happen whether or not $\overline{CE}$ is active. Thus **the Clock Read Buffer will not hold the same value indefinitely just because the TMS 9901 select bit is set.** The PSI will leave clock mode whenever the CPU reads to or writes from CRU bits 16-31, or if any device accesses a memory address with a 1 on the address line connected to S0 (A4 in a TMS 9900 system).

The logic controlling clock mode and the Clock Read Buffer may be illustrated as follows:



This logic summarizes our discussion above. There are two important things to note about clock mode and Clock Read Buffer update. First, you cannot inadvertently exit clock mode while you are reading the Clock Read Buffer, since you access it as CRU bits 1-14. Second, you cannot enter clock mode solely by accessing CRU bits 0-15; S0 changes clock mode only when the select bit is 1 (clock mode selected).

In order to read the most recent Clock Counter value, you must do two things:

- **Exit clock mode** so the Clock Read Buffer will receive the current Clock Counter contents.
- **Enter clock mode** so the Clock Read Buffer will be stable during the read itself.

Here is the appropriate instruction sequence:

```
LI      R12,PSI+1   LOAD PSI CRU BASE ADDRESS
SBZ     -1          EXIT CLOCK MODE TO UPDATE READ BUFFER
SBO     -1          ENTER CLOCK MODE TO STABILIZE READ BUFFER
STCR    R1,14       READ 14-BIT CLOCK READ BUFFER
```

TMS 9901 RESET LOGIC

You can reset a TMS 9901 in one of two ways:

1) **By inputting a low signal at $\overline{RST1}$.**
2) **By using a programmed reset via $\overline{RST2}$, a CRU bit.**

In order to use $\overline{RST1}$, a low level must be input at this pin for at least two clock periods.

You can reset the TMS 9901 under program control only when clock mode is selected (CRU bit 0 is 0). At this time, writing a 0 to CRU bit 15 ($\overline{RST2}$) causes the device to be reset. Thus, the following instruction sequence causes a TMS 9901 device reset:

```
LI      R12,PSI     LOAD PSI CRU BASE ADDRESS
SBO     0           ENTER CLOCK MODE
SBZ     15          RESET PSI
```

When the TMS 9901 is reset, the $\overline{INTREQ}$ signal is output high, IC0 through IC3 are output low, all interrupt requests are disabled, and all I/O CRU bits are placed in input mode.

DATA SHEETS

The following electrical specifications for the TMS 9900 and the TMS 9980A are out of date: we provide them here only to give a rough idea of timing and electrical requirements. Texas Instruments is revising its documentation on the TMS 9900 series parts. Revised data sheets will appear in updates to this volume and in the next edition.

TMS 9900 ELECTRICAL AND MECHANICAL SPECIFICATIONS

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	−65°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}.

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

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
		WE, 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
$C_{i(\phi1)}$	Clock-1 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi2)}$	Clock-2 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		150	200	pF
$C_{i(\phi3)}$	Clock-3 input capacitance		V_{BB} = −5, f = 1MHz, unmeasured pins at V_{SS}		100	150	pF
$C_{i(\phi4)}$	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

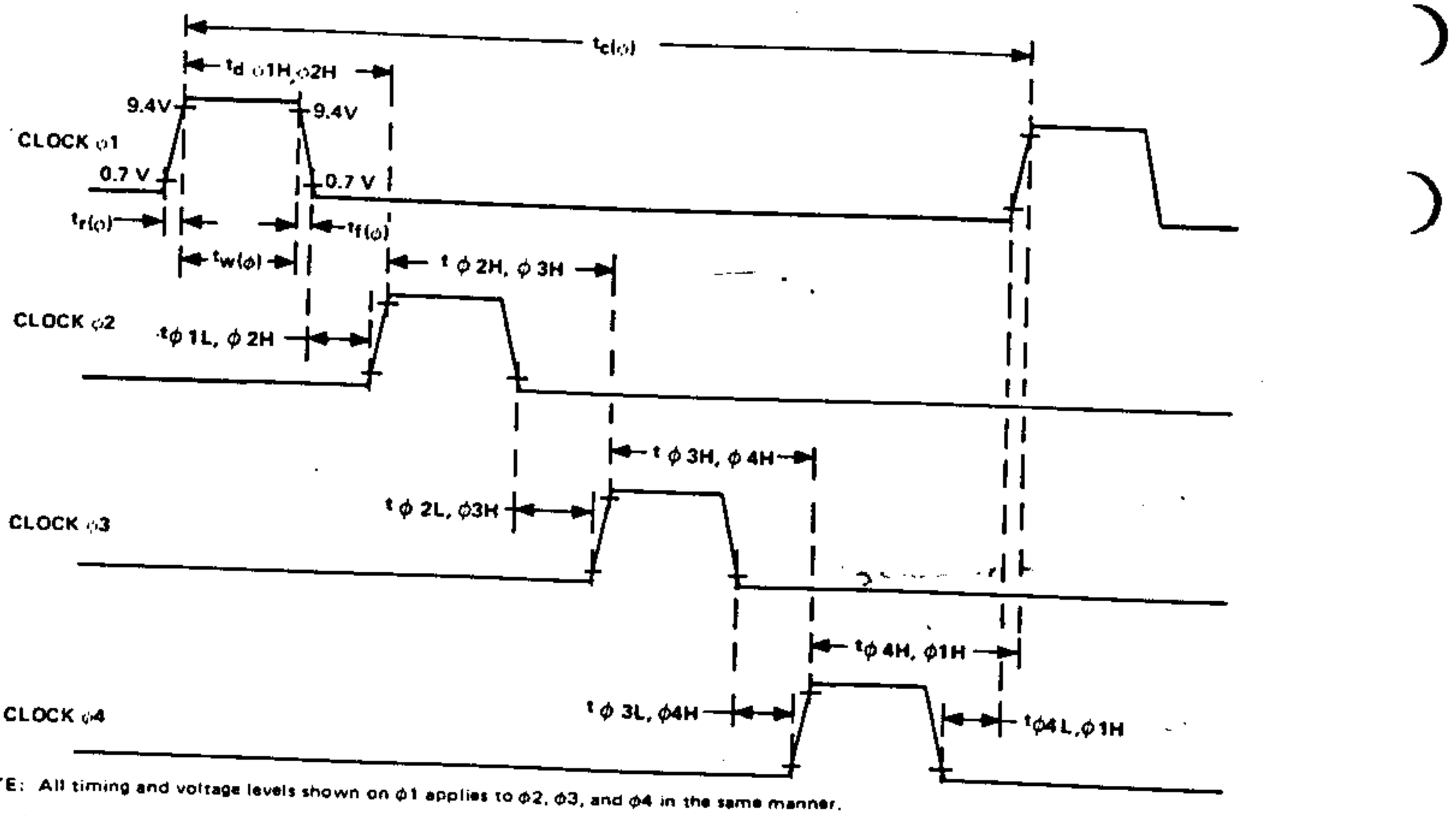
†All typical values are at T_A = 25°C and nominal voltages.
*D.C. Component of Operating Clock

TIMING REQUIREMENTS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURES 12 AND 13)

	PARAMETER	MIN	NOM	MAX	UNIT
$t_{c(\phi)}$	Clock cycle time	0.3	0.333	0.5	μs
$t_{r(\phi)}$	Clock rise time	5	12	25	ns
$t_{f(\phi)}$	Clock fall time	10	12	25	ns
$t_{w(\phi)}$	Pulse width, any clock high	40	45	100	ns
$t_{\phi 1L,\phi 2H}$	Delay time, clock 1 low to clock 2 high (time between clock pulses)	0	5		ns
$t_{\phi 2L,\phi 3H}$	Delay time, clock 2 low to clock 3 high (time between clock pulses)	0	5		ns
$t_{\phi 3L,\phi 4H}$	Delay time, clock 3 low to clock 4 high (time between clock pulses)	0	5		ns
$t_{\phi 4L,\phi 1H}$	Delay time, clock 4 low to clock 1 high (time between clock pulses)	0	5		ns
$t_{\phi 1H,\phi 2H}$	Delay time, clock 1 high to clock 2 high (time between leading edges)	73	80		ns
$t_{\phi 2H,\phi 3H}$	Delay time, clock 2 high to clock 3 high (time between leading edges)	73	80		ns
$t_{\phi 3H,\phi 4H}$	Delay time, clock 3 high to clock 4 high (time between leading edges)	73	80		ns
$t_{\phi 4H,\phi 1H}$	Delay time, clock 4 high to clock 1 high (time between leading edges)	73	80		ns
t_{su}	Data or control setup time before clock 1	30			ns
t_h	Data hold time after clock 1	10			ns

SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS (SEE FIGURE 13)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH} or t_{PHL} Propagation delay time, clocks to outputs	C_L = 200 pF		20	40	ns



NOTE: All timing and voltage levels shown on φ1 applies to φ2, φ3, and φ4 in the same manner.

FIGURE 12 – CLOCK TIMING

TMS 9980A/TMS 9981 ELECTRICAL AND MECHANICAL SPECIFICATIONS

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

Supply voltage, V_{CC} (see Note 1)	−0.3 to 15 V
Supply voltage, V_{DD} (see Note 1)	−0.3 to 15 V
Supply voltage, V_{BB} (see Note 1) (9980A only)	−5.25 to 0 V
All input voltages (see Note 1)	−0.3 to 15 V
Output voltage (see Note 1)	−2 V to 7 V
Continuous power dissipation	1.4 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 V_{SS}.

4.2 RECOMMENDED OPERATING CONDITIONS

	MIN	NOM	MAX	UNIT
Supply voltage, V_{BB} (9980A only)	−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}	2.2	2.4	V_{CC}+1	V
Low-level input voltage, V_{IL}	−1	0.4	0.8	V
Operating free-air temperature, T_A	0	20	70	°C

4.3 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}			±75	μA
		$\overline{WE}$, $\overline{MEMEN}$, $\overline{DBIN}$ during HOLDA	V_I = V_{SS} to V_{CC}			±75	
		Any other inputs	V_I = V_{SS} to V_{CC}			±10	
V_{OH}	High-level output voltage		I_O = −0.4 mA	2.4			V
V_{OL}	Low-level output voltage		I_O = 2 mA			0.5	V
			I_O = 3.2 mA			0.65	
I_{BB}	Supply current from V_{BB} (9980A Only)					1	mA
I_{CC}	Supply current from V_{CC}		0°C		50	60	mA
			70°C		40	50	
I_{DD}	Supply current from V_{DD}		0°C		70	80	mA
			70°C		65	75	
C_I	Input capacitance (any inputs except data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF
C_{DB}	Data bus capacitance		f = 1 MHz, unmeasured pins at V_{SS}		25		pF
C_O	Output capacitance (any output except data bus)		f = 1 MHz, unmeasured pins at V_{SS}		15		pF

*All typical values are at T_A = 25°C and nominal voltages.

CLOCK CHARACTERISTICS

The TMS 9980A and TMS 9981 have an internal 4-phase clock generator/driver. This is driven by an external TTL compatible signal to control the phase generation. In addition, the TMS 9981 provides an output (OSCOUT) that in conjunction with CKIN forms an on-chip crystal oscillator. This oscillator requires an external crystal and two capacitors as shown in Figure 13. The external signal or crystal must be 4 times the desired system frequency.

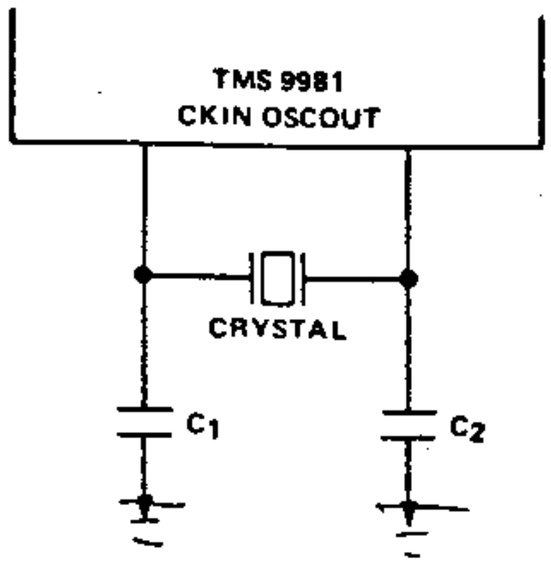


FIGURE 13 – CRYSTAL OSCILATOR CIRCUIT

Internal Crystal Oscillator (9981 Only)

The internal crystal oscillator is used as shown in Figure 13. The crystal should be a fundamental series resonant type. C_1 and C_2 represent the total capacitance on these pins including strays and parasitics.

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Crystal frequency	0°C-70°C	6		10	MHZ
C_1, C_2	0°C-70°C	10	15	25	pf

External Clock

The external clock on the TMS 9980A and optional on the TMS 9981, uses the CKIN pin. In this mode the OSCOUT pin of the TMS 9981 must be left floating. The external clock source must conform to the following specifications.

	PARAMETER	MIN	TYP	MAX	UNIT
f_{ext}	External source frequency*	6		10	MHz
V_H	External source high level	2.2			V
V_L	External source low level			0.8	V
T_r/T_f	External source rise/fall time		10		ns
T_{WH}	External source high level pulse width	40			ns
T_{WL}	External source low level pulse width	40			ns

*This allows a system speed of 1.5 MHz to 2 MHz.

SWITCHING CHARACTERISTICS OVER FULL RANGE OF RECOMMENDED OPERATING CONDITIONS

The timing of all the inputs and outputs are controlled by the internal 4 phase clock; thus all timings are based on the width of one phase of the internal clock. This is $1/f_{(CKIN)}$ (whether driven or from a crystal). This is also $\frac{1}{4}f_{system}$. In the following table this phase time is denoted t_w.

All external signals are with reference to $\phi 3$ (see Figure 14).

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
$t_r(\phi 3)$	Rise time of $\phi 3$	$t_w = 1/f_{(CKIN)} = \frac{1}{4}f_{system}$ C_L = 200pf	3	5	10	ns
$t_f(\phi 3)$	Fall time of $\phi 3$		5	7.5	15	ns
$t_w(\phi 3)$	Pulse width of $\phi 3$		t_w-15	t_w-10	t_w+10	ns
t_{su}	Data or control setup time*		t_w-30			ns
t_h	Data hold time*		$2t_w+10$			ns
$t_{PHL}(\overline{WE})$	Propagation delay time WE high to low		t_w-10	t_w	t_w+20	ns
$t_{PLH}(\overline{WE})$	Propagation delay time WE low to high		t_w	t_w+10	t_w+30	ns
t_{PHL}(CRUCLK)	Propagation delay time, CRUCLK high to low		−20	−10	+10	ns
t_{PLH}(CRUCLK)	Propagation delay time, CRUCLK low to high		$2t_w-10$	$2t_w$	$2t_w+20$	ns
t_{OV}	Delay time from output valid to $\phi 3$ low		t_w-50	t_w-30		ns
t_{OX}	Delay time from output invalid to $\phi 3$ low			t_w-20	t_w	ns

*All inputs except IC0-IC2 must be synchronized to meet these requirements. IC0-IC2 may change asynchronously. See section 2.10.4.

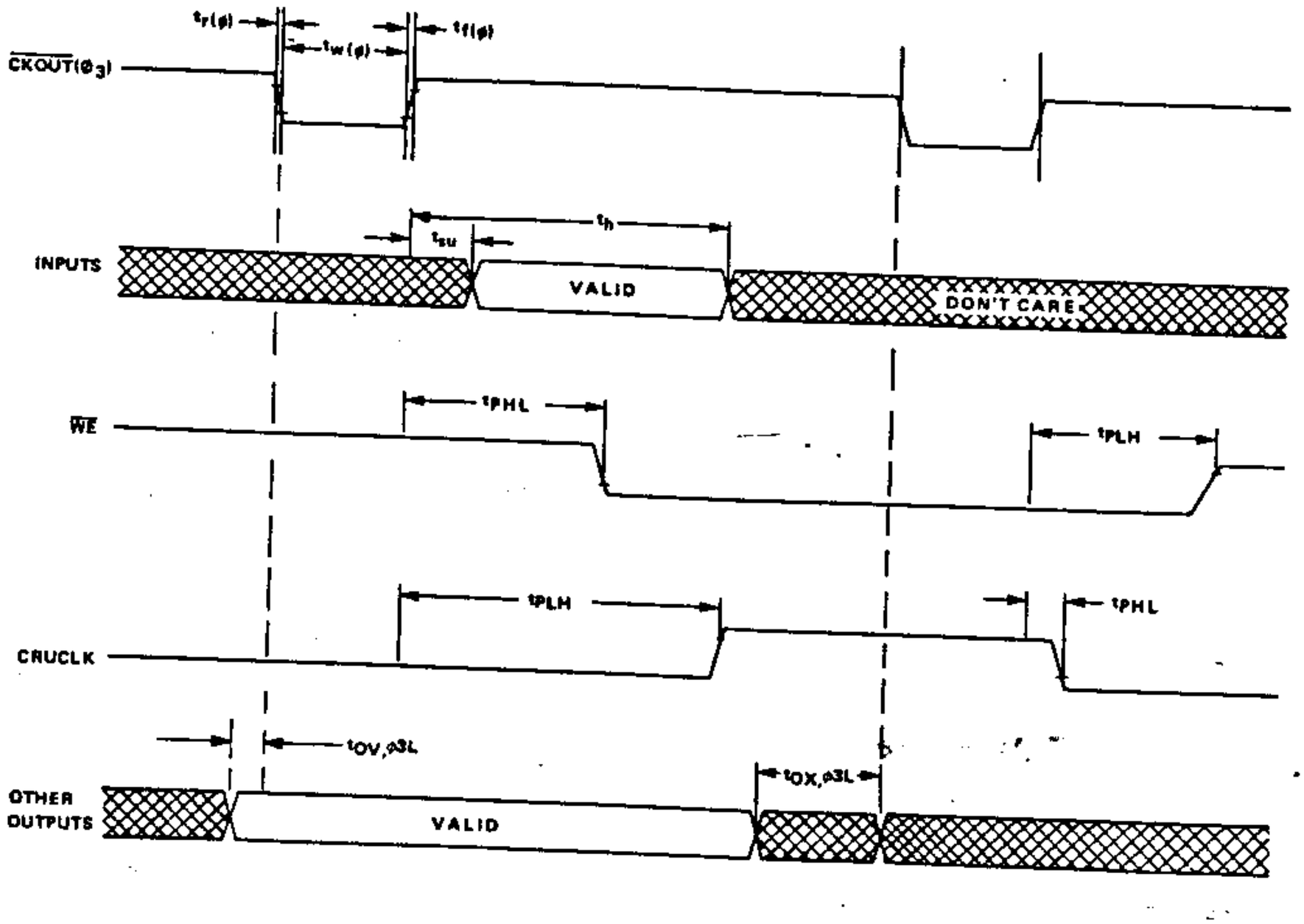


FIGURE 14 – EXTERNAL SIGNAL TIMING DIAGRAM