

**CHAPTER 2  
THE 8080 CENTRAL  
PROCESSOR UNIT**

The 8080 is a complete 8-bit parallel, central processor unit (CPU) for use in general purpose digital computer systems. It is fabricated on a single LSI chip (see Figure 2-1), using Intel's n-channel silicon gate MOS process. The 8080 transfers data and internal state information via an 8-bit, bidirectional 3-state Data Bus (D<sub>0</sub>-D<sub>7</sub>). Memory and peripheral device addresses are transmitted over a separate 16-

bit 3-state Address Bus (A<sub>0</sub>-A<sub>15</sub>). Six timing and control outputs (SYNC, DBIN, WAIT, WR, HLDA and INTE) emanate from the 8080, while four control inputs (READY, HOLD, INT and RESET), four power inputs (+12v, +5v, -5v, and GND) and two clock inputs ( $\phi_1$  and  $\phi_2$ ) are accepted by the 8080.

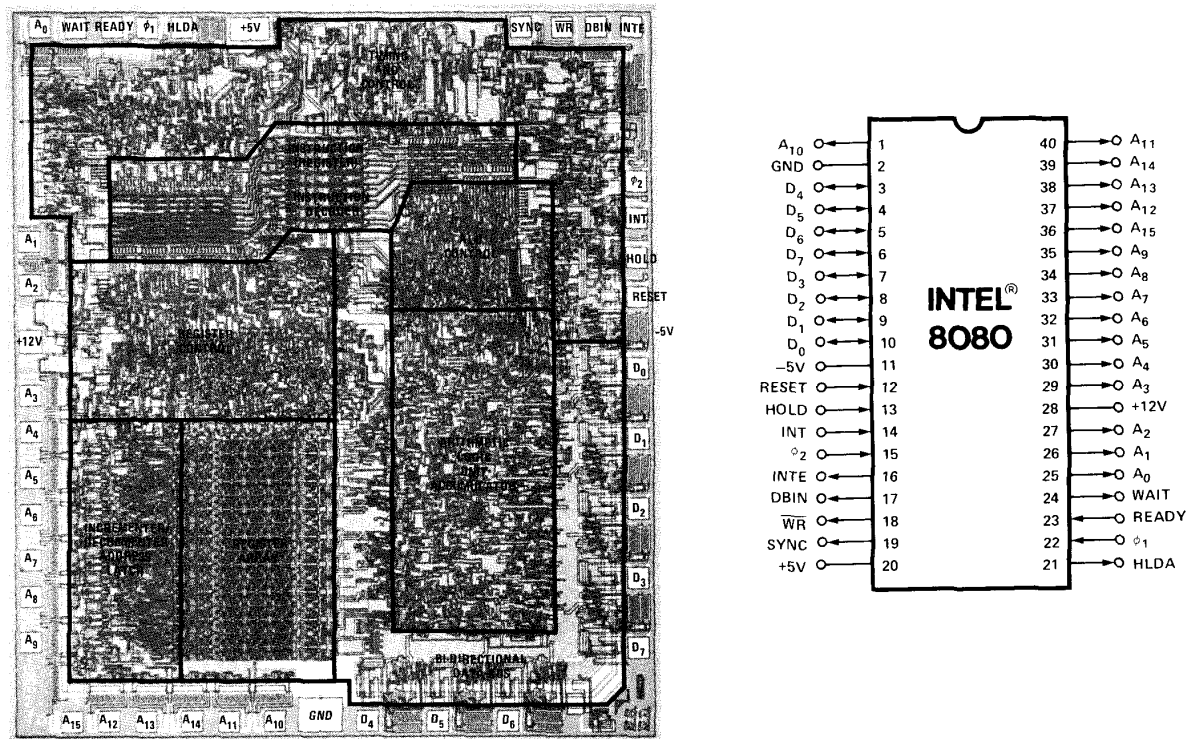


Figure 2-1. 8080 Photomicrograph With Pin Designations

## ARCHITECTURE OF THE 8080 CPU

The 8080 CPU consists of the following functional units:

- Register array and address logic
- Arithmetic and logic unit (ALU)
- Instruction register and control section
- Bi-directional, 3-state data bus buffer

Figure 2-2 illustrates the functional blocks within the 8080 CPU.

### Registers:

The register section consists of a static RAM array organized into six 16-bit registers:

- Program counter (PC)
- Stack pointer (SP)
- Six 8-bit general purpose registers arranged in pairs, referred to as B,C; D,E; and H,L
- A temporary register pair called W,Z

The program counter maintains the memory address of the current program instruction and is incremented auto-

matically during every instruction fetch. The stack pointer maintains the address of the next available stack location in memory. The stack pointer can be initialized to use any portion of read-write memory as a stack. The stack pointer is decremented when data is "pushed" onto the stack and incremented when data is "popped" off the stack (i.e., the stack grows "downward").

The six general purpose registers can be used either as single registers (8-bit) or as register pairs (16-bit). The temporary register pair, W,Z, is not program addressable and is only used for the internal execution of instructions.

Eight-bit data bytes can be transferred between the internal bus and the register array via the register-select multiplexer. Sixteen-bit transfers can proceed between the register array and the address latch or the incrementer/decrementer circuit. The address latch receives data from any of the three register pairs and drives the 16 address output buffers (A<sub>0</sub>-A<sub>15</sub>), as well as the incrementer/decrementer circuit. The incrementer/decrementer circuit receives data from the address latch and sends it to the register array. The 16-bit data can be incremented or decremented or simply transferred between registers.

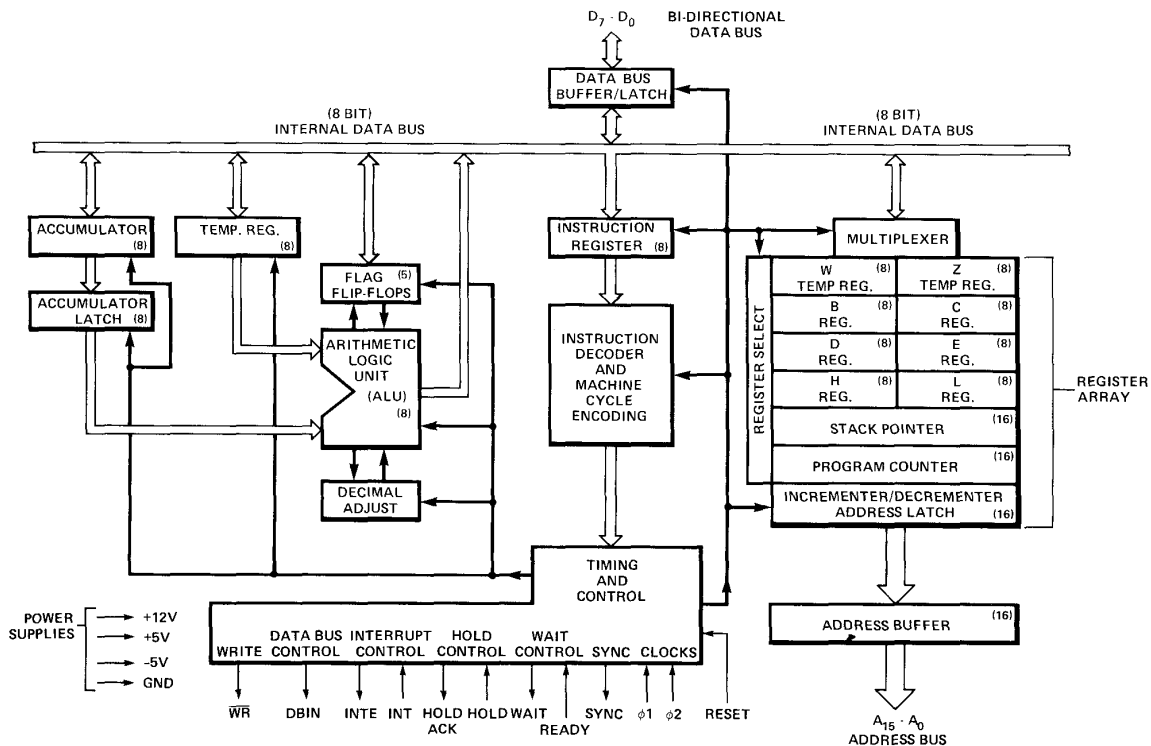


Figure 2-2. 8080 CPU Functional Block Diagram

## Arithmetic and Logic Unit (ALU):

The ALU contains the following registers:

- An 8-bit accumulator
- An 8-bit temporary accumulator (ACT)
- A 5-bit flag register: zero, carry, sign, parity and auxiliary carry
- An 8-bit temporary register (TMP)

Arithmetic, logical and rotate operations are performed in the ALU. The ALU is fed by the temporary register (TMP) and the temporary accumulator (ACT) and carry flip-flop. The result of the operation can be transferred to the internal bus or to the accumulator; the ALU also feeds the flag register.

The temporary register (TMP) receives information from the internal bus and can send all or portions of it to the ALU, the flag register and the internal bus.

The accumulator (ACC) can be loaded from the ALU and the internal bus and can transfer data to the temporary accumulator (ACT) and the internal bus. The contents of the accumulator (ACC) and the auxiliary carry flip-flop can be tested for decimal correction during the execution of the DAA instruction (see Chapter 4).

## Instruction Register and Control:

During an instruction fetch, the first byte of an instruction (containing the OP code) is transferred from the internal bus to the 8-bit instruction register.

The contents of the instruction register are, in turn, available to the instruction decoder. The output of the decoder, combined with various timing signals, provides the control signals for the register array, ALU and data buffer blocks. In addition, the outputs from the instruction decoder and external control signals feed the timing and state control section which generates the state and cycle timing signals.

## Data Bus Buffer:

This 8-bit bidirectional 3-state buffer is used to isolate the CPU's internal bus from the external data bus (D<sub>0</sub> through D<sub>7</sub>). In the output mode, the internal bus content is loaded into an 8-bit latch that, in turn, drives the data bus output buffers. The output buffers are switched off during input or non-transfer operations.

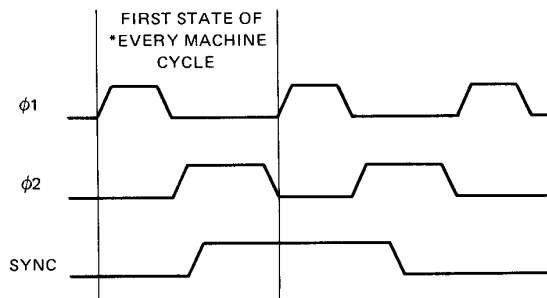
During the input mode, data from the external data bus is transferred to the internal bus. The internal bus is pre-charged at the beginning of each internal state, except for the transfer state (T<sub>3</sub>—described later in this chapter).

## THE PROCESSOR CYCLE

An **instruction cycle** is defined as the time required to fetch and execute an instruction. During the fetch, a selected instruction (one, two or three bytes) is extracted from memory and deposited in the CPU's instruction register. During the execution phase, the instruction is decoded and translated into specific processing activities.

Every instruction cycle consists of one, two, three, four or five machine cycles. A **machine cycle** is required each time the CPU accesses memory or an I/O port. The fetch portion of an instruction cycle requires one machine cycle for each byte to be fetched. The duration of the execution portion of the instruction cycle depends on the kind of instruction that has been fetched. Some instructions do not require any machine cycles other than those necessary to fetch the instruction; other instructions, however, require additional machine cycles to write or read data to/from memory or I/O devices. The DAD instruction is an exception in that it requires two additional machine cycles to complete an internal register-pair add (see Chapter 4).

Each machine cycle consists of three, four or five states. A state is the smallest unit of processing activity and is defined as the interval between two successive positive-going transitions of the  $\phi_1$  driven clock pulse. The 8080 is driven by a two-phase clock oscillator. All processing activities are referred to the period of this clock. The two non-overlapping clock pulses, labeled  $\phi_1$  and  $\phi_2$ , are furnished by external circuitry. It is the  $\phi_1$  clock pulse which divides each machine cycle into states. Timing logic within the 8080 uses the clock inputs to produce a SYNC pulse, which identifies the beginning of every machine cycle. The SYNC pulse is triggered by the low-to-high transition of  $\phi_2$ , as shown in Figure 2-3.



\*SYNC DOES NOT OCCUR IN THE SECOND AND THIRD MACHINE CYCLES OF A DAD INSTRUCTION SINCE THESE MACHINE CYCLES ARE USED FOR AN INTERNAL REGISTER-PAIR ADD.

Figure 2-3.  $\phi_1$ ,  $\phi_2$  And SYNC Timing

There are three exceptions to the defined duration of a state. They are the WAIT state, the hold (HLDA) state and the halt (HLTA) state, described later in this chapter. Because the WAIT, the HLDA, and the HLTA states depend upon external events, they are by their nature of indeterminate length. Even these exceptional states, however, must

be synchronized with the pulses of the driving clock. Thus, the duration of all states are integral multiples of the clock period.

To summarize then, each **clock period** marks a **state**; three to five states constitute a machine cycle; and one to five **machine cycles** comprise an **instruction cycle**. A full instruction cycle requires anywhere from four to eight-teen states for its completion, depending on the kind of instruction involved.

### Machine Cycle Identification:

With the exception of the DAD instruction, there is just one consideration that determines how many machine cycles are required in any given instruction cycle: the number of times that the processor must reference a memory address or an addressable peripheral device, in order to fetch and execute the instruction. Like many processors, the 8080 is so constructed that it can transmit only one address per machine cycle. Thus, if the fetch and execution of an instruction requires two memory references, then the instruction cycle associated with that instruction consists of two machine cycles. If five such references are called for, then the instruction cycle contains five machine cycles.

Every instruction cycle has at least one reference to memory, during which the instruction is fetched. An instruction cycle must always have a fetch, even if the execution of the instruction requires no further references to memory. The first machine cycle in every instruction cycle is therefore a FETCH. Beyond that, there are no fast rules. It depends on the kind of instruction that is fetched.

Consider some examples. The add-register (ADD r) instruction is an instruction that requires only a single machine cycle (FETCH) for its completion. In this one-byte instruction, the contents of one of the CPU's six general purpose registers is added to the existing contents of the accumulator. Since all the information necessary to execute the command is contained in the eight bits of the instruction code, only one memory reference is necessary. Three states are used to extract the instruction from memory, and one additional state is used to accomplish the desired addition. The entire instruction cycle thus requires only one machine cycle that consists of four states, or four periods of the external clock.

Suppose now, however, that we wish to add the contents of a specific memory location to the existing contents of the accumulator (ADD M). Although this is quite similar in principle to the example just cited, several additional steps will be used. An extra machine cycle will be used, in order to address the desired memory location.

The actual sequence is as follows. First the processor extracts from memory the one-byte instruction word addressed by its program counter. This takes three states. The eight-bit instruction word obtained during the FETCH machine cycle is deposited in the CPU's instruction register and used to direct activities during the remainder of the instruction cycle. Next, the processor sends out, as an address,

the contents of its H and L registers. The eight-bit data word returned during this MEMORY READ machine cycle is placed in a temporary register inside the 8080 CPU. By now three more clock periods (states) have elapsed. In the seventh and final state, the contents of the temporary register are added to those of the accumulator. Two machine cycles, consisting of seven states in all, complete the "ADD M" instruction cycle.

At the opposite extreme is the save H and L registers (SHLD) instruction, which requires five machine cycles. During an "SHLD" instruction cycle, the contents of the processor's H and L registers are deposited in two sequentially adjacent memory locations; the destination is indicated by two address bytes which are stored in the two memory locations immediately following the operation code byte. The following sequence of events occurs:

- (1) A FETCH machine cycle, consisting of four states. During the first three states of this machine cycle, the processor fetches the instruction indicated by its program counter. The program counter is then incremented. The fourth state is used for internal instruction decoding.
- (2) A MEMORY READ machine cycle, consisting of three states. During this machine cycle, the byte indicated by the program counter is read from memory and placed in the processor's Z register. The program counter is incremented again.
- (3) Another MEMORY READ machine cycle, consisting of three states, in which the byte indicated by the processor's program counter is read from memory and placed in the W register. The program counter is incremented, in anticipation of the next instruction fetch.
- (4) A MEMORY WRITE machine cycle, of three states, in which the contents of the L register are transferred to the memory location pointed to by the present contents of the W and Z registers. The state following the transfer is used to increment the W,Z register pair so that it indicates the next memory location to receive data.
- (5) A MEMORY WRITE machine cycle, of three states, in which the contents of the H register are transferred to the new memory location pointed to by the W,Z register pair.

In summary, the "SHLD" instruction cycle contains five machine cycles and takes 16 states to execute.

Most instructions fall somewhere between the extremes typified by the "ADD r" and the "SHLD" instructions. The input (INP) and the output (OUT) instructions, for example, require three machine cycles: a FETCH, to obtain the instruction; a MEMORY READ, to obtain the address of the object peripheral; and an INPUT or an OUTPUT machine cycle, to complete the transfer.

While no one instruction cycle will consist of more than five machine cycles, the following ten different types of machine cycles may occur within an instruction cycle:

- (1) FETCH (M1)
- (2) MEMORY READ
- (3) MEMORY WRITE
- (4) STACK READ
- (5) STACK WRITE
- (6) INPUT
- (7) OUTPUT
- (8) INTERRUPT
- (9) HALT
- (10) HALT • INTERRUPT

The machine cycles that actually do occur in a particular instruction cycle depend upon the kind of instruction, with the overriding stipulation that the first machine cycle in any instruction cycle is always a FETCH.

The processor identifies the machine cycle in progress by transmitting an eight-bit status word during the first state of every machine cycle. Updated status information is presented on the 8080's data lines (D<sub>0</sub>-D<sub>7</sub>), during the SYNC interval. This data should be saved in latches, and used to develop control signals for external circuitry. Table 2-1 shows how the positive-true status information is distributed on the processor's data bus.

Status signals are provided principally for the control of external circuitry. Simplicity of interface, rather than machine cycle identification, dictates the logical definition of individual status bits. You will therefore observe that certain processor machine cycles are uniquely identified by a single status bit, but that others are not. The M<sub>1</sub> status bit (D<sub>6</sub>), for example, unambiguously identifies a FETCH machine cycle. A STACK READ, on the other hand, is indicated by the coincidence of STACK and MEMR signals. Machine cycle identification data is also valuable in the test and de-bugging phases of system development. Table 2-1 lists the status bit outputs for each type of machine cycle.

### State Transition Sequence:

Every machine cycle within an instruction cycle consists of three to five active states (referred to as T<sub>1</sub>, T<sub>2</sub>, T<sub>3</sub>, T<sub>4</sub>, T<sub>5</sub> or T<sub>W</sub>). The actual number of states depends upon the instruction being executed, and on the particular machine cycle within the greater instruction cycle. The state transition diagram in Figure 2-4 shows how the 8080 proceeds from state to state in the course of a machine cycle. The diagram also shows how the READY, HOLD, and INTERRUPT lines are sampled during the machine cycle, and how the conditions on these lines may modify the

basic transition sequence. In the present discussion, we are concerned only with the basic sequence and with the READY function. The HOLD and INTERRUPT functions will be discussed later.

The 8080 CPU does not directly indicate its internal state by transmitting a "state control" output during each state; instead, the 8080 supplies direct control output (INTE, HLDA, DBIN,  $\overline{WR}$  and WAIT) for use by external circuitry.

Recall that the 8080 passes through at least three states in every machine cycle, with each state defined by successive low-to-high transitions of the  $\phi_1$  clock. Figure 2-5 shows the timing relationships in a typical FETCH machine cycle. Events that occur in each state are referenced to transitions of the  $\phi_1$  and  $\phi_2$  clock pulses.

The SYNC signal identifies the first state (T<sub>1</sub>) in every machine cycle. As shown in Figure 2-5, the SYNC signal is related to the leading edge of the  $\phi_2$  clock. There is a delay ( $t_{DC}$ ) between the low-to-high transition of  $\phi_2$  and the positive-going edge of the SYNC pulse. There also is a corresponding delay (also  $t_{DC}$ ) between the next  $\phi_2$  pulse and the falling edge of the SYNC signal. Status information is displayed on D<sub>0</sub>-D<sub>7</sub> during the same  $\phi_2$  to  $\phi_2$  interval. Switching of the status signals is likewise controlled by  $\phi_2$ .

The rising edge of  $\phi_2$  during T<sub>1</sub> also loads the processor's address lines (A<sub>0</sub>-A<sub>15</sub>). These lines become stable within a brief delay ( $t_{DA}$ ) of the  $\phi_2$  clocking pulse, and they remain stable until the first  $\phi_2$  pulse after state T<sub>3</sub>. This gives the processor ample time to read the data returned from memory.

Once the processor has sent an address to memory, there is an opportunity for the memory to request a WAIT. This it does by pulling the processor's READY line low, prior to the "Ready set-up" interval ( $t_{RS}$ ) which occurs during the  $\phi_2$  pulse within state T<sub>2</sub> or T<sub>W</sub>. As long as the READY line remains low, the processor will idle, giving the memory time to respond to the addressed data request. Refer to Figure 2-5.

The processor responds to a wait request by entering an alternative state (T<sub>W</sub>) at the end of T<sub>2</sub>, rather than proceeding directly to the T<sub>3</sub> state. Entry into the T<sub>W</sub> state is indicated by a WAIT signal from the processor, acknowledging the memory's request. A low-to-high transition on the WAIT line is triggered by the rising edge of the  $\phi_1$  clock and occurs within a brief delay ( $t_{DC}$ ) of the actual entry into the T<sub>W</sub> state.

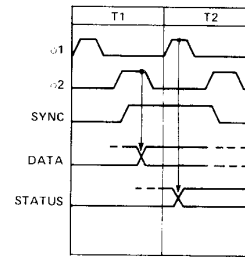
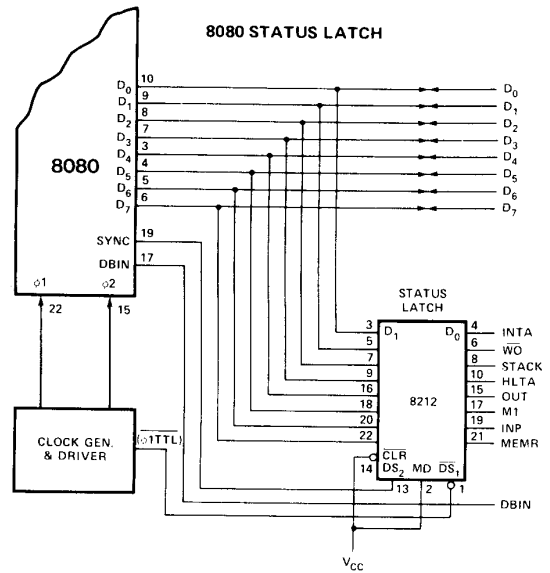
A wait period may be of indefinite duration. The processor remains in the waiting condition until its READY line again goes high. A READY indication **must** precede the falling edge of the  $\phi_2$  clock by a specified interval ( $t_{RS}$ ), in order to guarantee an exit from the T<sub>W</sub> state. The cycle may then proceed, beginning with the rising edge of the next  $\phi_1$  clock. A WAIT interval will therefore consist of an integral number of T<sub>W</sub> states and will always be a multiple of the clock period.

Instructions for the 8080 require from one to five machine cycles for complete execution. The 8080 sends out 8 bit of status information on the data bus at the beginning of each machine cycle (during SYNC time). The following table defines the status information.

**STATUS INFORMATION DEFINITION**

Symbols	Data Bus Bit	Definition
INTA*	D <sub>0</sub>	Acknowledge signal for INTERRUPT request. Signal should be used to gate a re-start instruction onto the data bus when DBIN is active.
$\overline{WO}$	D <sub>1</sub>	Indicates that the operation in the current machine cycle will be a WRITE memory or OUTPUT function ( $\overline{WO} = 0$ ). Otherwise, a READ memory or INPUT operation will be executed.
STACK	D <sub>2</sub>	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
HLTA	D <sub>3</sub>	Acknowledge signal for HALT instruction.
OUT	D <sub>4</sub>	Indicates that the address bus contains the address of an output device and the data bus will contain the output data when $\overline{WR}$ is active.
M <sub>1</sub>	D <sub>5</sub>	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
INP*	D <sub>6</sub>	Indicates that the address bus contains the address of an input device and the input data should be placed on the data bus when DBIN is active.
MEMR*	D <sub>7</sub>	Designates that the data bus will be used for memory read data.

\*These three status bits can be used to control the flow of data onto the 8080 data bus.



**STATUS WORD CHART**

DATA BUS BIT	STATUS INFORMATION	TYPE OF MACHINE CYCLE									
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D <sub>0</sub>	INTA	0	0	0	0	0	0	0	1	0	1
D <sub>1</sub>	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D <sub>2</sub>	STACK	0	0	0	1	1	0	0	0	0	0
D <sub>3</sub>	HLTA	0	0	0	0	0	0	0	0	1	1
D <sub>4</sub>	OUT	0	0	0	0	0	0	1	0	0	0
D <sub>5</sub>	M <sub>1</sub>	1	0	0	0	0	0	0	1	0	1
D <sub>6</sub>	INP	0	0	0	0	0	1	0	0	0	0
D <sub>7</sub>	MEMR	1	1	0	1	0	0	0	0	1	0

Table 2-1. 8080 Status Bit Definitions

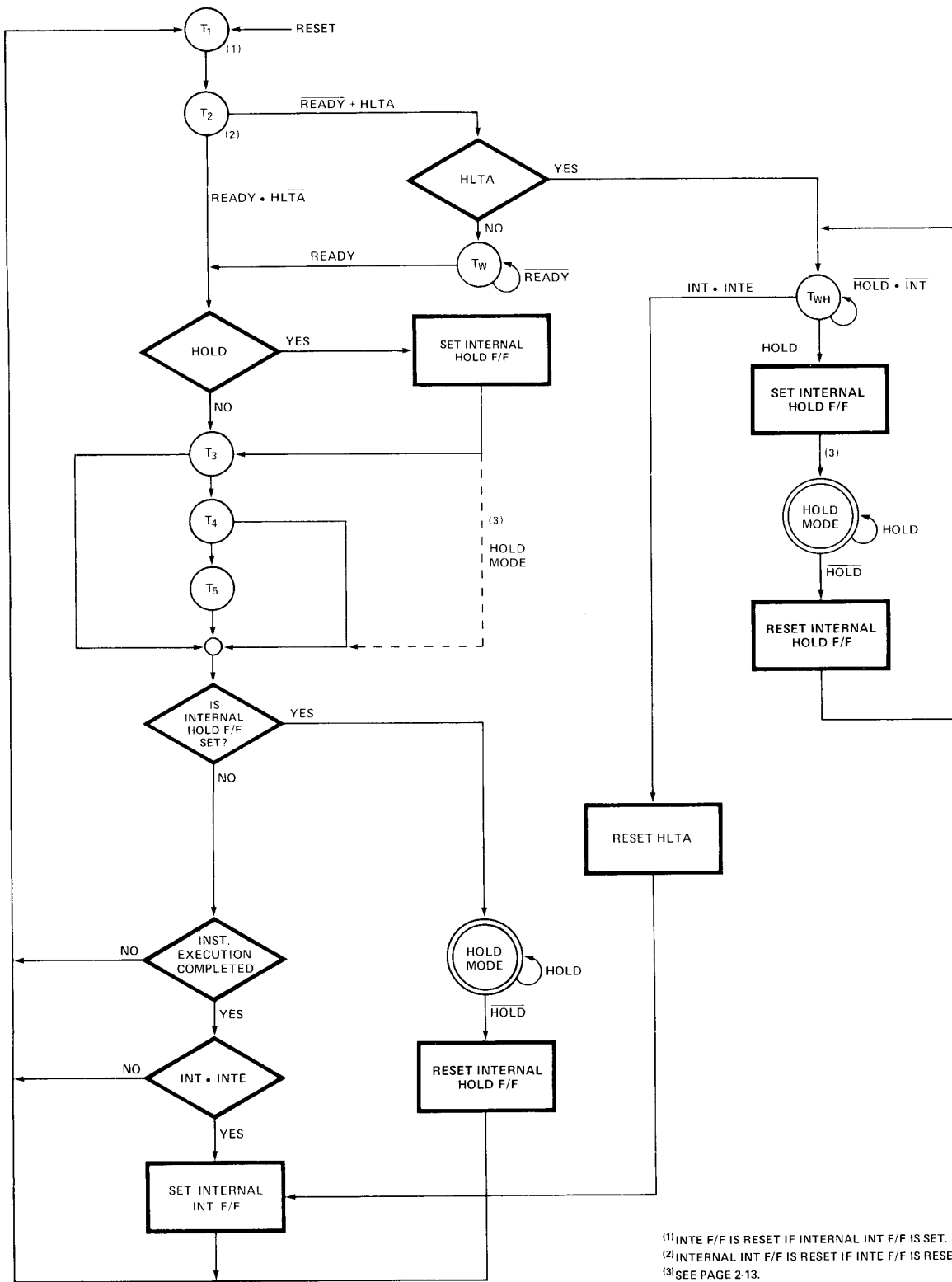


Figure 2-4. CPU State Transition Diagram

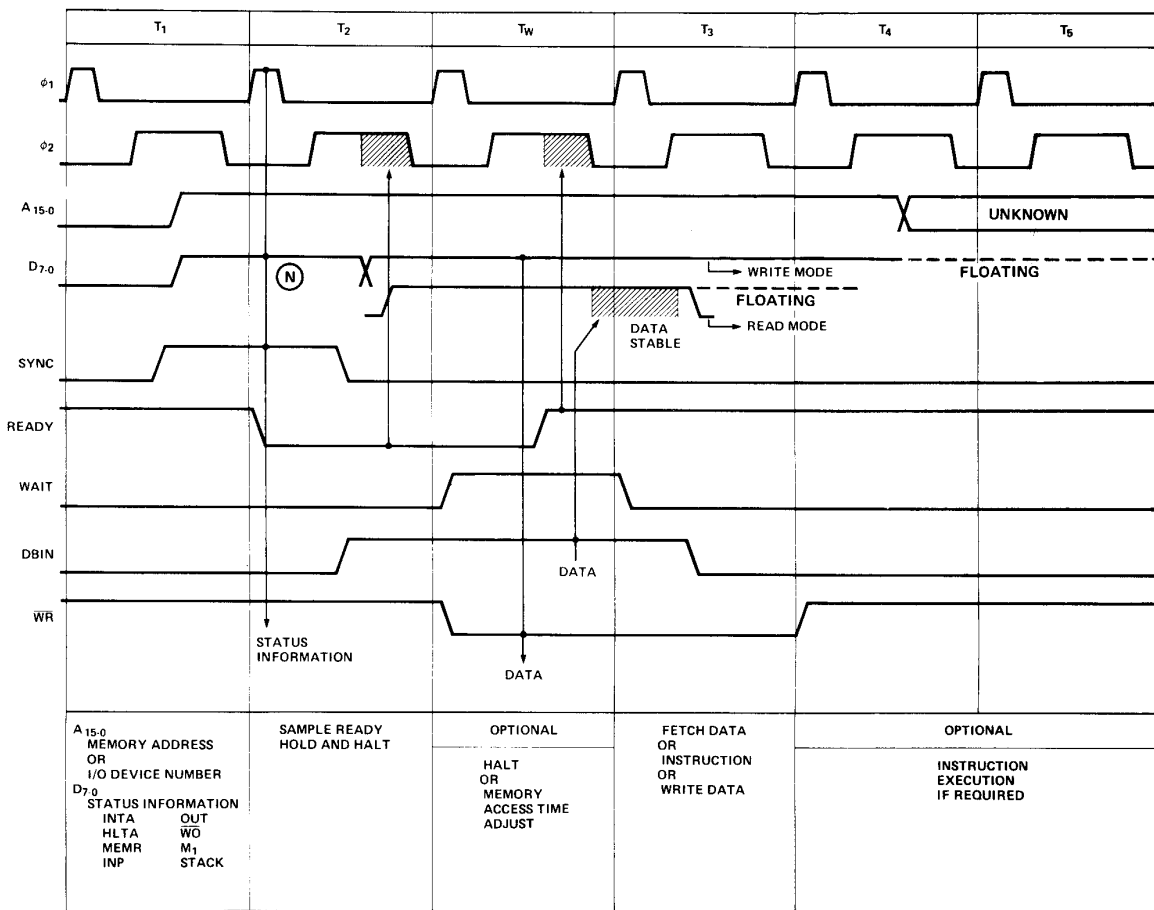
The events that take place during the T<sub>3</sub> state are determined by the kind of machine cycle in progress. In a FETCH machine cycle, the processor interprets the data on its data bus as an instruction. During a MEMORY READ or a STACK READ, data on this bus is interpreted as a data word. The processor outputs data on this bus during a MEMORY WRITE machine cycle. During I/O operations, the processor may either transmit or receive data, depending on whether an OUTPUT or an INPUT operation is involved.

Figure 2-6 illustrates the timing that is characteristic of a data input operation. As shown, the low-to-high transition of  $\phi_2$  during T<sub>2</sub> clears status information from the processor's data lines, preparing these lines for the receipt of incoming data. The data presented to the processor must have stabilized prior to both the " $\phi_1$ -data set-up" interval ( $t_{DS1}$ ), that precedes the falling edge of the  $\phi_1$  pulse defining state T<sub>3</sub>, and the " $\phi_2$ -data set-up" interval ( $t_{DS2}$ ), that precedes the rising edge of  $\phi_2$  in state T<sub>3</sub>. This same

data must remain stable during the "data hold" interval ( $t_{DH}$ ) that occurs following the rising edge of the  $\phi_2$  pulse. Data placed on these lines by memory or by other external devices will be sampled during T<sub>3</sub>.

During the input of data to the processor, the 8080 generates a DBIN signal which should be used externally to enable the transfer. Machine cycles in which DBIN is available include: FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of  $\phi_2$  during state T<sub>2</sub> and terminated by the corresponding edge of  $\phi_2$  during T<sub>3</sub>. Any T<sub>W</sub> phases intervening between T<sub>2</sub> and T<sub>3</sub> will therefore extend DBIN by one or more clock periods.

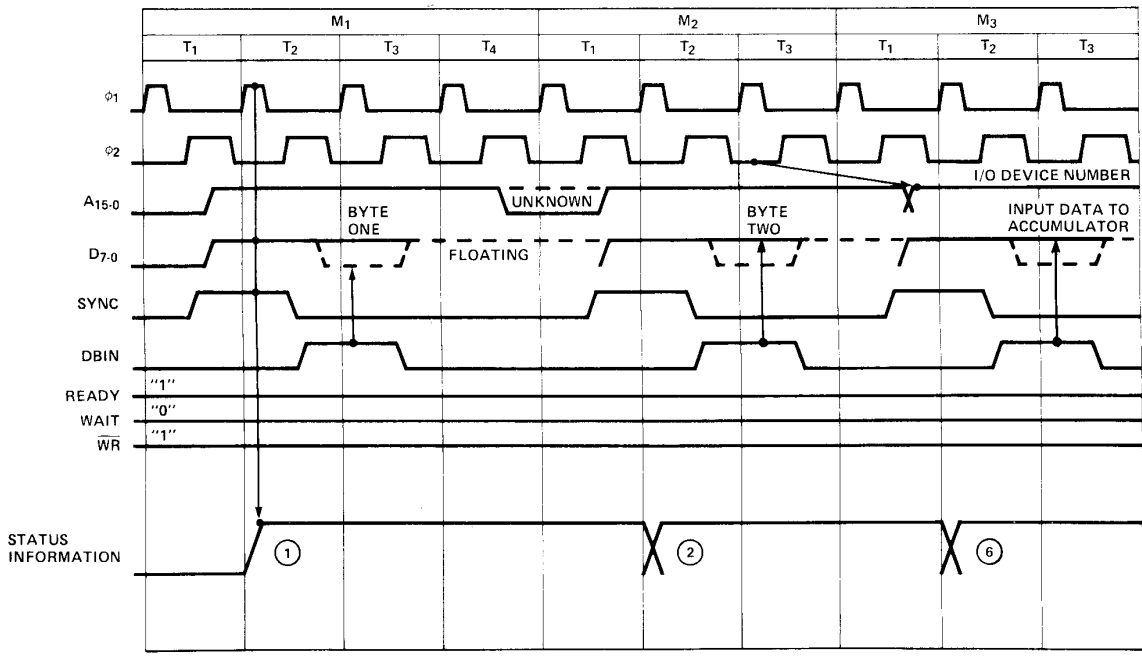
Figure 2-7 shows the timing of a machine cycle in which the processor outputs data. Output data may be destined either for memory or for peripherals. The rising edge of  $\phi_2$  within state T<sub>2</sub> clears status information from the CPU's data lines, and loads in the data which is to be output to external devices. This substitution takes place within the



NOTE: (N) Refer to Status Word Chart on Page 2-6.

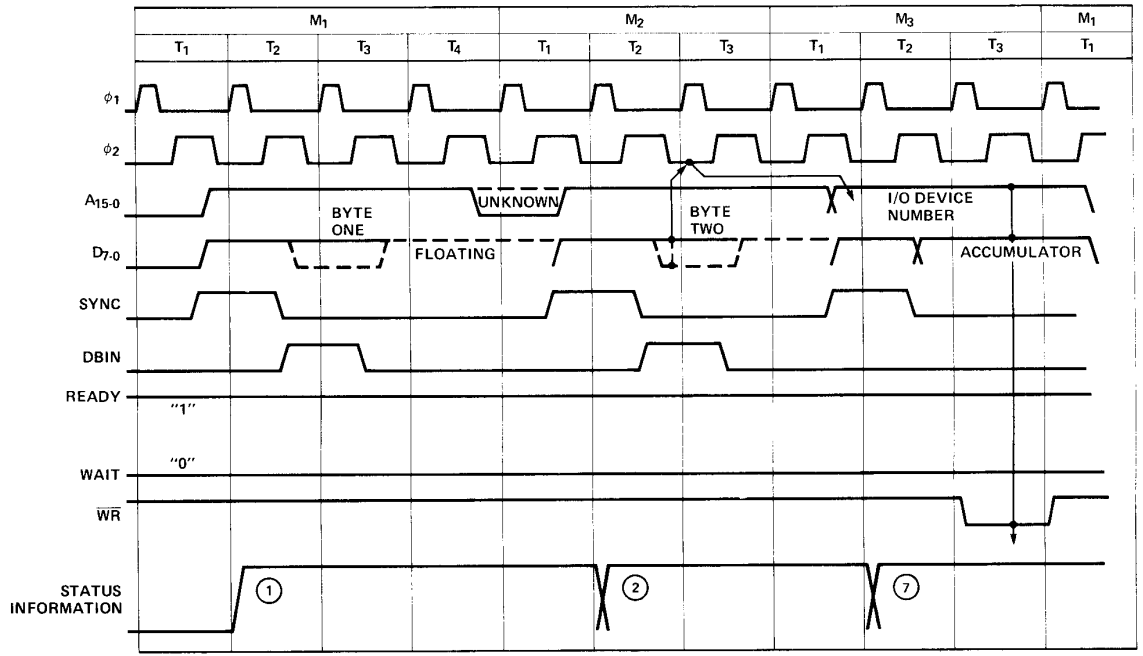
Figure 2-5. Basic 8080 Instruction Cycle





NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-6. Input Instruction Cycle



NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-7. Output Instruction Cycle

“data output delay” interval ( $t_{DD}$ ) following the  $\phi_2$  clock’s leading edge. Data on the bus remains stable throughout the remainder of the machine cycle, until replaced by updated status information in the subsequent  $T_1$  state. Observe that a READY signal is necessary for completion of an OUTPUT machine cycle. Unless such an indication is present, the processor enters the  $T_W$  state, following the  $T_2$  state. Data on the output lines remains stable in the interim, and the processing cycle will not proceed until the READY line again goes high.

The 8080 CPU generates a  $\overline{WR}$  output for the synchronization of external transfers, during those machine cycles in which the processor outputs data. These include MEMORY WRITE, STACK WRITE, and OUTPUT. The negative-going leading edge of  $\overline{WR}$  is referenced to the rising edge of the first  $\phi_1$  clock pulse following  $T_2$ , and occurs within a brief delay ( $t_{DC}$ ) of that event.  $\overline{WR}$  remains low until re-triggered by the leading edge of  $\phi_1$  during the state following  $T_3$ . Note that any  $T_W$  states intervening between  $T_2$  and  $T_3$  of the output machine cycle will neces-

sarily extend  $\overline{WR}$ , in much the same way that  $\overline{DBIN}$  is affected during data input operations.

All processor machine cycles consist of at least three states:  $T_1$ ,  $T_2$ , and  $T_3$  as just described. If the processor has to wait for a response from the peripheral or memory with which it is communicating, then the machine cycle may also contain one or more  $T_W$  states. During the three basic states, data is transferred to or from the processor.

After the  $T_3$  state, however, it becomes difficult to generalize.  $T_4$  and  $T_5$  states are available, if the execution of a particular instruction requires them. But not all machine cycles make use of these states. It depends upon the kind of instruction being executed, and on the particular machine cycle within the instruction cycle. The processor will terminate any machine cycle as soon as its processing activities are completed, rather than proceeding through the  $T_4$  and  $T_5$  states every time. Thus the 8080 may exit a machine cycle following the  $T_3$ , the  $T_4$ , or the  $T_5$  state and proceed directly to the  $T_1$  state of the next machine cycle.

STATE	ASSOCIATED ACTIVITIES
$T_1$	A memory address or I/O device number is placed on the Address Bus (A <sub>15-0</sub> ); status information is placed on Data Bus (D <sub>7-0</sub> ).
$T_2$	The CPU samples the READY and HOLD inputs and checks for halt instruction.
$T_W$ (optional)	Processor enters wait state if READY is low or if HALT instruction has been executed.
$T_3$	An instruction byte (FETCH machine cycle), data byte (MEMORY READ, STACK READ) or interrupt instruction (INTERRUPT machine cycle) is input to the CPU from the Data Bus; or a data byte (MEMORY WRITE, STACK WRITE or OUTPUT machine cycle) is output onto the data bus.
$T_4$ $T_5$ (optional)	States $T_4$ and $T_5$ are available if the execution of a particular instruction requires them; if not, the CPU may skip one or both of them. $T_4$ and $T_5$ are only used for internal processor operations.

Table 2-2. State Definitions

## INTERRUPT SEQUENCES

The 8080 has the built-in capacity to handle external interrupt requests. A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high.

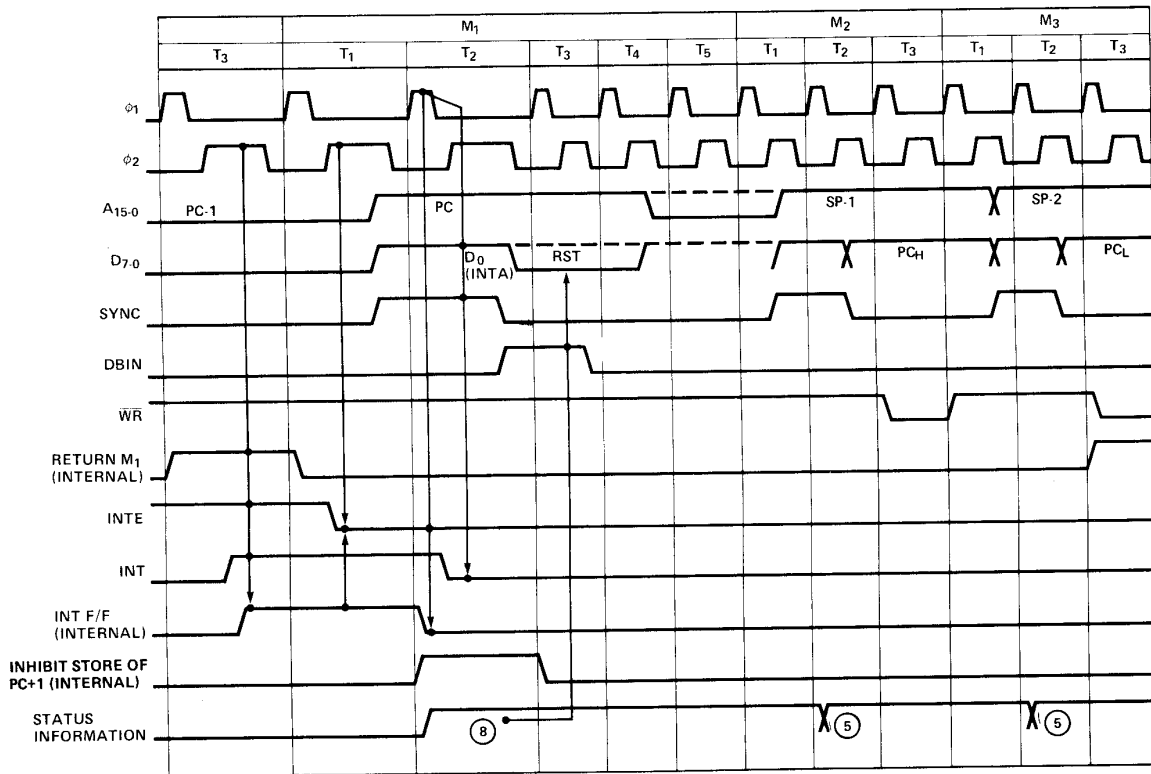
The interrupt (INT) input is asynchronous, and a request may therefore originate at any time during any instruction cycle. Internal logic re-clocks the external request, so that a proper correspondence with the driving clock is established. As Figure 2-8 shows, an interrupt request (INT) arriving during the time that the interrupt enable line (INTE) is high, acts in coincidence with the  $\phi_2$  clock to set the internal interrupt latch. This event takes place during the last state of the instruction cycle in which the request occurs, thus ensuring that any instruction in progress is completed before the interrupt can be processed.

The INTERRUPT machine cycle which follows the arrival of an enabled interrupt request resembles an ordinary FETCH machine cycle in most respects. The  $M_1$  status bit is transmitted as usual during the SYNC interval. It is accompanied, however, by an INTA status bit ( $D_0$ ) which acknowledges the external request. The contents of the program counter are latched onto the CPU's address lines during  $T_1$ , but the counter itself is not incremented during the INTERRUPT machine cycle, as it otherwise would be.

In this way, the pre-interrupt status of the program counter is preserved, so that data in the counter may be restored by the interrupted program after the interrupt request has been processed.

The interrupt cycle is otherwise indistinguishable from an ordinary FETCH machine cycle. The processor itself takes no further special action. It is the responsibility of the peripheral logic to see that an eight-bit interrupt instruction is "jammed" onto the processor's data bus during state  $T_3$ . In a typical system, this means that the data-in bus from memory must be temporarily disconnected from the processor's main data bus, so that the interrupting device can command the main bus without interference.

The 8080's instruction set provides a special one-byte call which facilitates the processing of interrupts (the ordinary program Call takes three bytes). This is the RESTART instruction (RST). A variable three-bit field embedded in the eight-bit field of the RST enables the interrupting device to direct a Call to one of eight fixed memory locations. The decimal addresses of these dedicated locations are: 0, 8, 16, 24, 32, 40, 48, and 56. Any of these addresses may be used to store the first instruction(s) of a routine designed to service the requirements of an interrupting device. Since the (RST) is a call, completion of the instruction also stores the old program counter contents on the STACK.



NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-8. Interrupt Timing

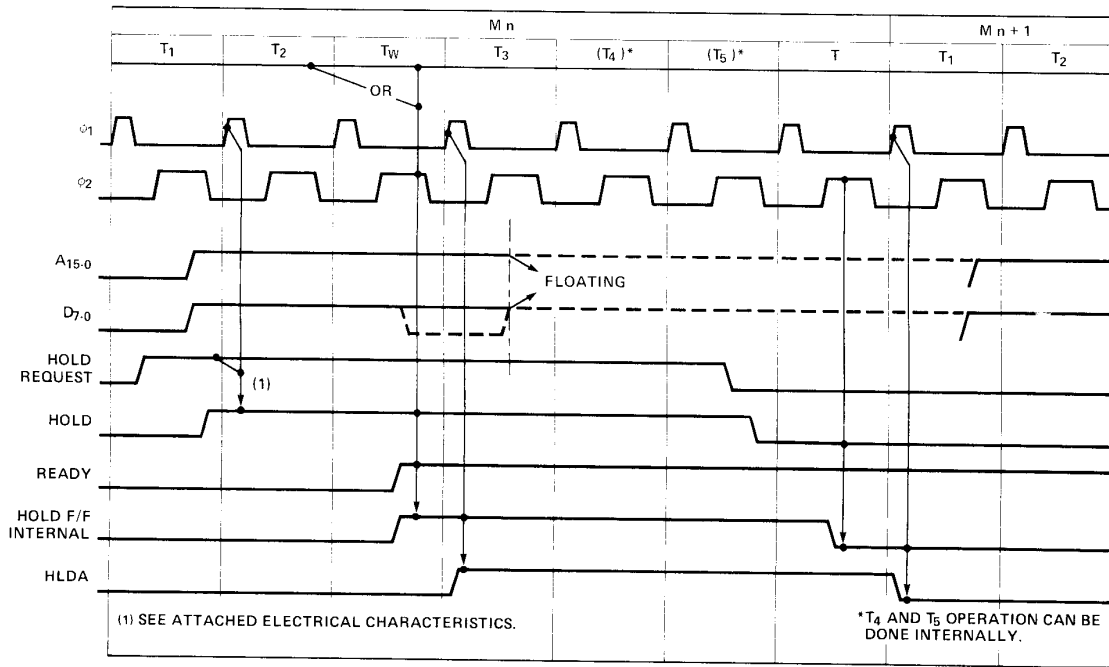


Figure 2-9. HOLD Operation (Read Mode)

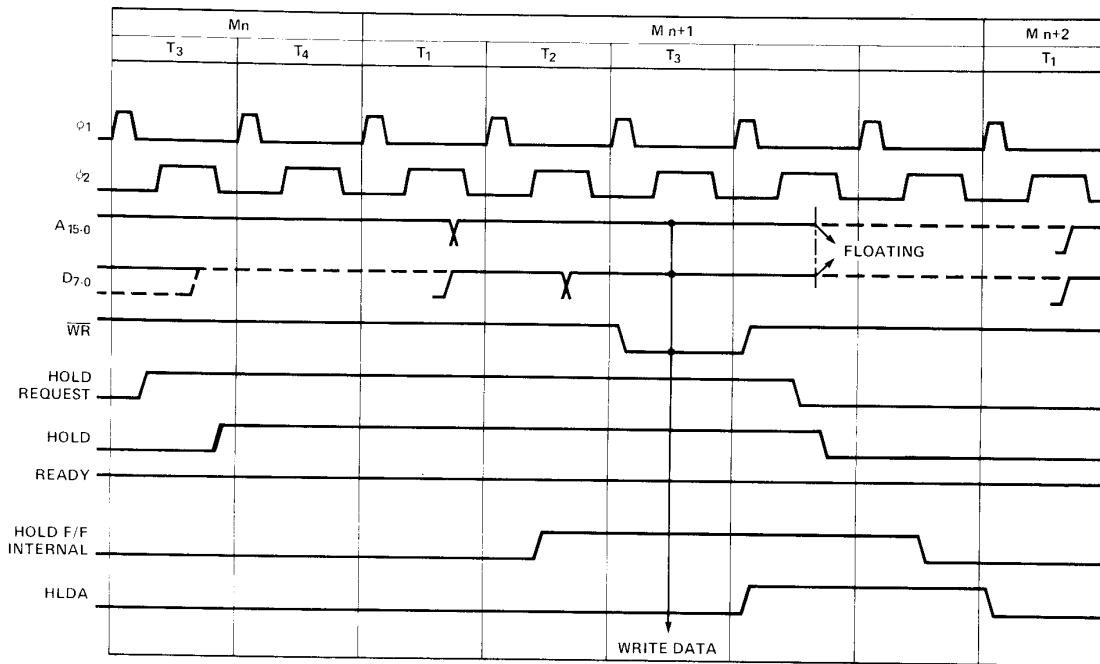


Figure 2-10. HOLD Operation (Write Mode)

## HOLD SEQUENCES

The 8080A CPU contains provisions for Direct Memory Access (DMA) operations. By applying a HOLD to the appropriate control pin on the processor, an external device can cause the CPU to suspend its normal operations and relinquish control of the address and data busses. The processor responds to a request of this kind by floating its address to other devices sharing the busses. At the same time, the processor acknowledges the HOLD by placing a high on its HLDA output pin. During an acknowledged HOLD, the address and data busses are under control of the peripheral which originated the request, enabling it to conduct memory transfers without processor intervention.

Like the interrupt, the HOLD input is synchronized internally. A HOLD signal must be stable prior to the "Hold set-up" interval ( $t_{HS}$ ), that precedes the rising edge of  $\phi_2$ .

Figures 2-9 and 2-10 illustrate the timing involved in HOLD operations. Note the delay between the asynchronous HOLD REQUEST and the re-clocked HOLD. As shown in the diagram, a coincidence of the READY, the HOLD, and the  $\phi_2$  clocks sets the internal hold latch. Setting the latch enables the subsequent rising edge of the  $\phi_1$  clock pulse to trigger the HLDA output.

Acknowledgement of the HOLD REQUEST precedes slightly the actual floating of the processor's address and data lines. The processor acknowledges a HOLD at the beginning of  $T_3$ , if a read or an input machine cycle is in progress (see Figure 2-9). Otherwise, acknowledgement is deferred until the beginning of the state following  $T_3$  (see Figure 2-10). In both cases, however, the HLDA goes high within a specified delay ( $t_{DC}$ ) of the rising edge of the selected  $\phi_1$  clock pulse. Address and data lines are floated within a brief delay after the rising edge of the next  $\phi_2$  clock pulse. This relationship is also shown in the diagrams.

To all outward appearances, the processor has suspended its operations once the address and data busses are floated. Internally, however, certain functions may continue. If a HOLD REQUEST is acknowledged at  $T_3$ , and if the processor is in the middle of a machine cycle which requires four or more states to complete, the CPU proceeds through  $T_4$  and  $T_5$  before coming to a rest. Not until the end of the machine cycle is reached will processing activities cease. Internal processing is thus permitted to overlap the external DMA transfer, improving both the efficiency and the speed of the entire system.

The processor exits the holding state through a sequence similar to that by which it entered. A HOLD REQUEST is terminated asynchronously when the external device has completed its data transfer. The HLDA output

returns to a low level following the leading edge of the next  $\phi_1$  clock pulse. Normal processing resumes with the machine cycle following the last cycle that was executed.

## HALT SEQUENCES

When a halt instruction (HLT) is executed, the CPU enters the halt state ( $T_{WH}$ ) after state  $T_2$  of the next machine cycle, as shown in Figure 2-11. There are only three ways in which the 8080 can exit the halt state:

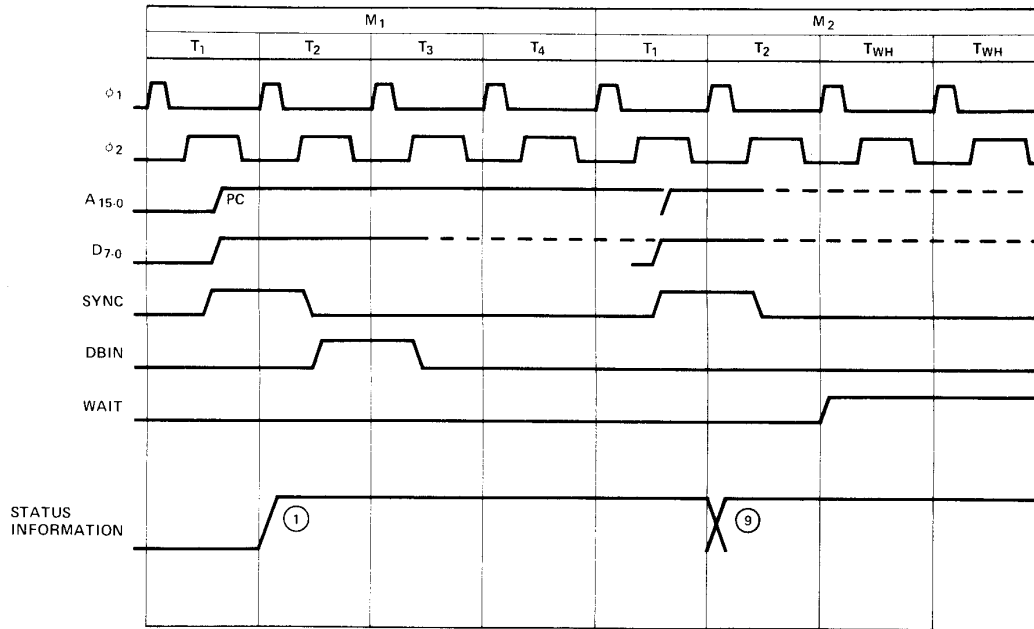
- A high on the RESET line will always reset the 8080 to state  $T_1$ ; RESET also clears the program counter.
- A HOLD input will cause the 8080 to enter the hold state, as previously described. When the HOLD line goes low, the 8080 re-enters the halt state on the rising edge of the next  $\phi_1$  clock pulse.
- An interrupt (i.e., INT goes high while INTE is enabled) will cause the 8080 to exit the Halt state and enter state  $T_1$  on the rising edge of the next  $\phi_1$  clock pulse. NOTE: The interrupt enable (INTE) flag **must** be set when the halt state is entered; otherwise, the 8080 will only be able to exit via a RESET signal.

Figure 2-12 illustrates halt sequencing in flow chart form.

## START-UP OF THE 8080 CPU

When power is applied initially to the 8080, the processor begins operating immediately. The contents of its program counter, stack pointer, and the other working registers are naturally subject to random factors and cannot be specified. For this reason, it will be necessary to begin the power-up sequence with RESET.

An external RESET signal of three clock period duration (minimum) restores the processor's internal program counter to zero. Program execution thus begins with memory location zero, following a RESET. Systems which require the processor to wait for an explicit start-up signal will store a halt instruction (EI, HLT) in the first two locations. A manual or an automatic INTERRUPT will be used for starting. In other systems, the processor may begin executing its stored program immediately. Note, however, that the RESET has no effect on status flags, or on any of the processor's working registers (accumulator, registers, or stack pointer). The contents of these registers remain indeterminate, until initialized explicitly by the program.



NOTE: (N) Refer to Status Word Chart on Page 2-6

Figure 2-11. HALT Timing

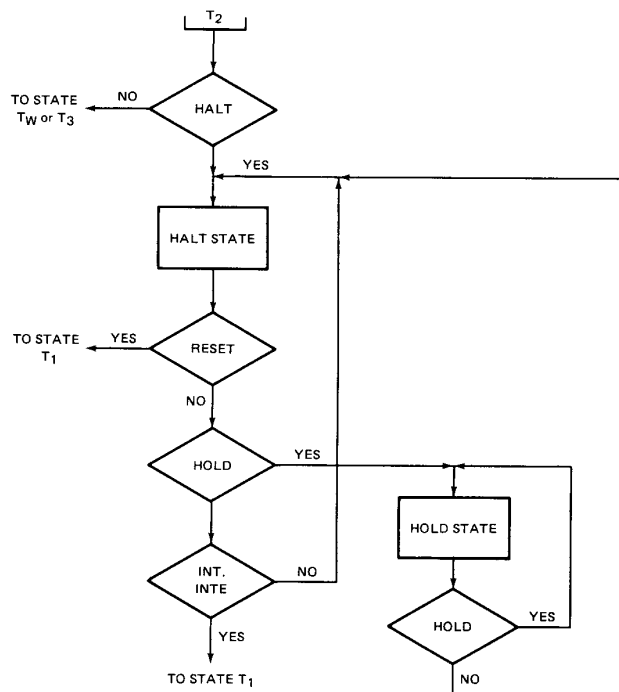
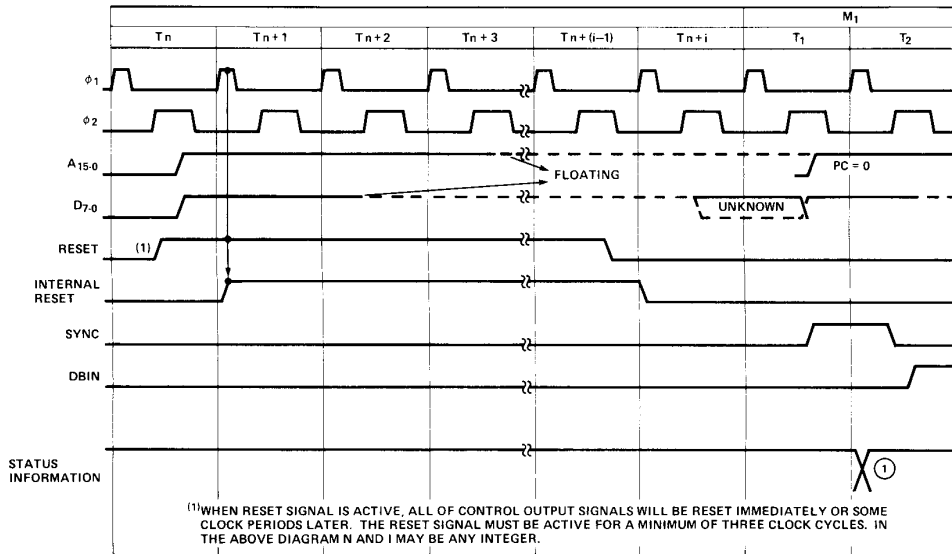
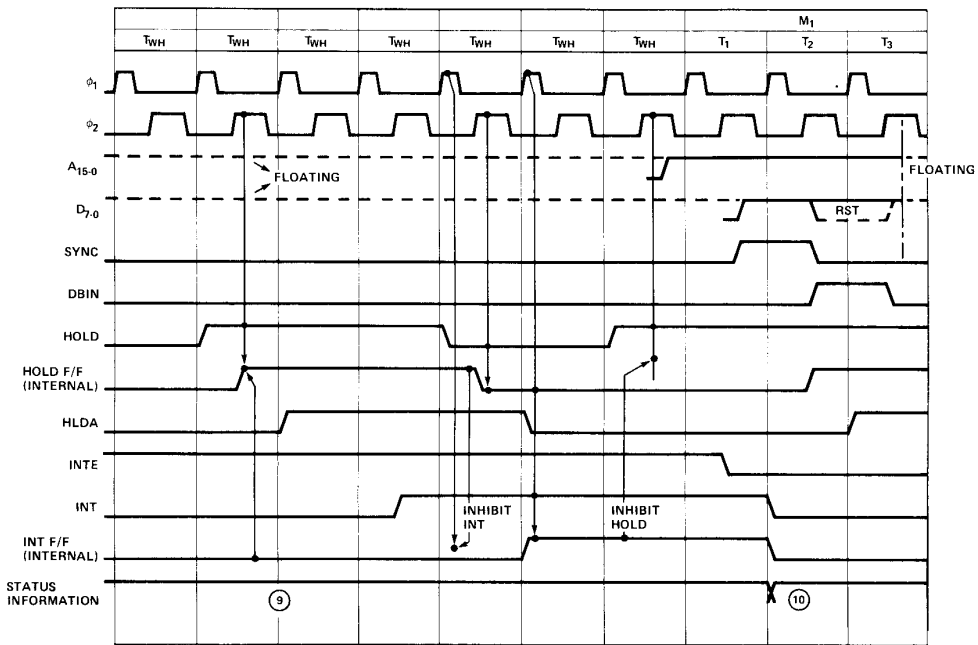


Figure 2-12. HALT Sequence Flow Chart.



NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-13. Reset.



NOTE: (N) Refer to Status Word Chart on Page 2-6.

Figure 2-14. Relation between HOLD and INT in the HALT State.

MNEMONIC	OP CODE		M1[1]					M2		
	D <sub>7</sub> D <sub>6</sub> D <sub>5</sub> D <sub>4</sub>	D <sub>3</sub> D <sub>2</sub> D <sub>1</sub> D <sub>0</sub>	T1	T2[2]	T3	T4	T5	T1	T2[2]	T3
MOV r <sub>1</sub> ,r <sub>2</sub>	0 1 D D	D S S S	PC OUT STATUS	PC = PC + 1	INST→TMP/IR	(SSS)→TMP	(TMP)→DDD			
MOV r, M	0 1 D D	D 1 1 0				X[3]		HL OUT STATUS[6]	DATA	→DDD
MOV M, r	0 1 1 1	0 S S S				(SSS)→TMP		HL OUT STATUS[7]	(TMP)	→DATA BUS
SPHL	1 1 1 1	1 0 0 1				(HL) → SP				
MVI r, data	0 0 D D	D 1 1 0				X		PC OUT STATUS[6]	B2	→DDDD
MVI M, data	0 0 1 1	0 1 1 0				X			B2	→TMP
LXI rp, data	0 0 R P	0 0 0 1				X			PC = PC + 1	B2 → r1
LDA addr	0 0 1 1	1 0 1 0				X			PC = PC + 1	B2 → Z
STA addr	0 0 1 1	0 0 1 0				X			PC = PC + 1	B2 → Z
LHLD addr	0 0 1 0	1 0 1 0				X			PC = PC + 1	B2 → Z
SHLD addr	0 0 1 0	0 0 1 0				X		PC OUT STATUS[6]	PC = PC + 1	B2 → Z
LDAX rp[4]	0 0 R P	1 0 1 0				X		rp OUT STATUS[6]	DATA	→A
STAX rp[4]	0 0 R P	0 0 1 0				X		rp OUT STATUS[7]	(A)	→DATA BUS
XCHG	1 1 1 0	1 0 1 1				(HL) ↔ (DE)				
ADD r	1 0 0 0	0 S S S				(SSS)→TMP (A)→ACT		[9]	(ACT)+(TMP)→A	
ADD M	1 0 0 0	0 1 1 0				(A)→ACT		HL OUT STATUS[6]	DATA	→TMP
ADI data	1 1 0 0	0 1 1 0				(A)→ACT		PC OUT STATUS[6]	PC = PC + 1	B2 → TMP
ADC r	1 0 0 0	1 S S S				(SSS)→TMP (A)→ACT		[9]	(ACT)+(TMP)+CY→A	
ADC M	1 0 0 0	1 1 1 0				(A)→ACT		HL OUT STATUS[6]	DATA	→TMP
ACI data	1 1 0 0	1 1 1 0				(A)→ACT		PC OUT STATUS[6]	PC = PC + 1	B2 → TMP
SUB r	1 0 0 1	0 S S S				(SSS)→TMP (A)→ACT		[9]	(ACT)-(TMP)→A	
SUB M	1 0 0 1	0 1 1 0				(A)→ACT		HL OUT STATUS[6]	DATA	→TMP
SUI data	1 1 0 1	0 1 1 0				(A)→ACT		PC OUT STATUS[6]	PC = PC + 1	B2 → TMP
SBB r	1 0 0 1	1 S S S				(SSS)→TMP (A)→ACT		[9]	(ACT)-(TMP)-CY→A	
SBB M	1 0 0 1	1 1 1 0				(A)→ACT		HL OUT STATUS[6]	DATA	→TMP
SBI data	1 1 0 1	1 1 1 0				(A)→ACT		PC OUT STATUS[6]	PC = PC + 1	B2 → TMP
INR r	0 0 D D	D 1 0 0				(DDD)→TMP (TMP) + 1→ALU	ALU→DDD			
INR M	0 0 1 1	0 1 0 0				X		HL OUT STATUS[6]	DATA (TMP)+1	→TMP →ALU
DCR r	0 0 D D	D 1 0 1				(DDD)→TMP (TMP)+1→ALU	ALU→DDD			
DCR M	0 0 1 1	0 1 0 1				X		HL OUT STATUS[6]	DATA (TMP)-1	→TMP →ALU
INX rp	0 0 R P	0 0 1 1				(RP) + 1 → RP				
DCX rp	0 0 R P	1 0 1 1				(RP) - 1 → RP				
DAD rp[8]	0 0 R P	1 0 0 1				X		(r) <sub>i</sub> →ACT	(L)→TMP, (ACT)+(TMP)→ALU	ALU→L, CY
DAA	0 0 1 0	0 1 1 1				DAA→A, FLAGS[10]				
ANA r	1 0 1 0	0 S S S				(SSS)→TMP (A)→ACT		[9]	(ACT)+(TMP)→A	
ANA M	1 0 1 0	0 1 1 0	PC OUT STATUS	PC = PC + 1	INST→TMP/IR	(A)→ACT		HL OUT STATUS[6]	DATA	→TMP



M3			M4			M5				
T1	T2[2]	T3	T1	T2[2]	T3	T1	T2[2]	T3	T4	T5
HL OUT STATUS[7]	(TMP)	→ DATA BUS								
PC OUT STATUS[6]	PC = PC + 1	B3 → rh								
	PC = PC + 1	B3 → W	WZ OUT STATUS[6]	DATA →	A					
	PC = PC + 1	B3 → W	WZ OUT STATUS[7]	(A) →	DATA BUS					
	PC = PC + 1	B3 → W	WZ OUT STATUS[6]	DATA →	L	WZ OUT STATUS[6]	DATA →	H		
PC OUT STATUS[6]	PC = PC + 1	B3 → W	WZ OUT STATUS[7]	(L) →	DATA BUS	WZ OUT STATUS[7]	(H) →	DATA BUS		
[9]	(ACT)+(TMP)→A									
[9]	(ACT)+(TMP)→A									
[9]	(ACT)+(TMP)+CY→A									
[9]	(ACT)+(TMP)+CY→A									
[9]	(ACT)-(TMP)→A									
[9]	(ACT)-(TMP)→A									
[9]	(ACT)-(TMP)-CY→A									
[9]	(ACT)-(TMP)-CY→A									
HL OUT STATUS[7]		ALU → DATA BUS								
HL OUT STATUS[7]		ALU → DATA BUS								
(rh)→ACT	(H)→TMP (ACT)+(TMP)+CY→ALU	ALU→H, CY								
[9]	(ACT)+(TMP)→A									

MNEMONIC	OP CODE		M1 <sup>[1]</sup>					M2		
	D <sub>7</sub> D <sub>6</sub> D <sub>5</sub> D <sub>4</sub>	D <sub>3</sub> D <sub>2</sub> D <sub>1</sub> D <sub>0</sub>	T1	T2 <sup>[2]</sup>	T3	T4	T5	T1	T2 <sup>[2]</sup>	T3
ANI data	1 1 1 0	0 1 1 0	PC OUT STATUS	PC = PC + 1	INST→TMP/IR	(A)→ACT		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→TMP
XRA r	1 0 1 0	1 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)+(TPM)→A	
XRA M	1 0 1 0	1 1 1 0				(A)→ACT		HL OUT STATUS <sub>[6]</sub>	DATA	→TMP
XRI data	1 1 1 0	1 1 1 0				(A)→ACT		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→TMP
ORA r	1 0 1 1	0 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)+(TPM)→A	
ORA M	1 0 1 1	0 1 1 0				(A)→ACT		HL OUT STATUS <sub>[6]</sub>	DATA	→TMP
ORI data	1 1 1 1	0 1 1 0				(A)→ACT		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→TMP
CMP r	1 0 1 1	1 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)-(TMP), FLAGS	
CMP M	1 0 1 1	1 1 1 0				(A)→ACT		HL OUT STATUS <sub>[6]</sub>	DATA	→TMP
CPI data	1 1 1 1	1 1 1 0				(A)→ACT		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→TMP
RLC	0 0 0 0	0 1 1 1				(A)→ALU ROTATE		[9]	ALU→A, CY	
RRC	0 0 0 0	1 1 1 1				(A)→ALU ROTATE		[9]	ALU→A, CY	
RAL	0 0 0 1	0 1 1 1				(A, CY)→ALU ROTATE		[9]	ALU→A, CY	
RAR	0 0 0 1	1 1 1 1				(A, CY)→ALU ROTATE		[9]	ALU→A, CY	
CMA	0 0 1 0	1 1 1 1				(A)→A				
CMC	0 0 1 1	1 1 1 1				CY→CY				
STC	0 0 1 1	0 1 1 1				1→CY				
JMP addr	1 1 0 0	0 0 1 1					X	PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z
J cond addr <sup>[17]</sup>	1 1 C C	C 0 1 0				JUDGE CONDITION		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z
CALL addr	1 1 0 0	1 1 0 1				SP = SP - 1		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z
C cond addr <sup>[17]</sup>	1 1 C C	C 1 0 0				JUDGE CONDITION IF TRUE, SP = SP - 1		PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z
RET	1 1 0 0	1 0 0 1					X	SP OUT STATUS <sub>[15]</sub>	SP = SP + 1 DATA	→Z
R cond addr <sup>[17]</sup>	1 1 C C	C 0 0 0				INST→TMP/IR	JUDGE CONDITION <sup>[14]</sup>	SP OUT STATUS <sub>[15]</sub>	SP = SP + 1 DATA	→Z
RST n	1 1 N N	N 1 1 1			φ→W INST→TMP/IR		SP = SP - 1	SP OUT STATUS <sub>[16]</sub>	SP = SP - 1 (PCH)	→DATA BUS
PCHL	1 1 1 0	1 0 0 1			INST→TMP/IR	(HL) → PC				
PUSH rp	1 1 R P	0 1 0 1					SP = SP - 1	SP OUT STATUS <sub>[16]</sub>	SP = SP - 1 (rh)	→DATA BUS
PUSH PSW	1 1 1 1	0 1 0 1					SP = SP - 1	SP OUT STATUS <sub>[16]</sub>	SP = SP - 1 (A)	→DATA BUS
POP rp	1 1 R P	0 0 0 1					X	SP OUT STATUS <sub>[15]</sub>	SP = SP + 1 DATA	→r1
POP PSW	1 1 1 1	0 0 0 1					X	SP OUT STATUS <sub>[15]</sub>	SP = SP + 1 DATA	→FLAGS
XTHL	1 1 1 0	0 0 1 1					X	SP OUT STATUS <sub>[15]</sub>	SP = SP + 1 DATA	→Z
IN port	1 1 0 1	1 0 1 1					X	PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z, W
OUT port	1 1 0 1	0 0 1 1					X	PC OUT STATUS <sub>[6]</sub>	PC = PC + 1 B2	→Z, W
EI	1 1 1 1	1 0 1 1				SET INTE F/F				
DI	1 1 1 1	0 0 1 1				RESET INTE F/F				
HLT	0 1 1 1	0 1 1 0					X	PC OUT STATUS	HALT MODE <sup>[20]</sup>	
NOP	0 0 0 0	0 0 0 0	PC OUT STATUS	PC = PC + 1	INST→TMP/IR		X			

M3			M4			M5					
T1	T2[2]	T3	T1	T2[2]	T3	T1	T2[2]	T3	T4	T5	
[9]	(ACT)+(TMP)→A										
[9]	(ACT)+(TMP)→A										
[9]	(ACT)+(TMP)→A										
[9]	(ACT)+(TMP)→A										
[9]	(ACT)+(TMP)→A										
[9]	(ACT)-(TMP); FLAGS										
[9]	(ACT)-(TMP); FLAGS										
PC OUT STATUS[6]	PC = PC + 1 B3 → W									WZ OUT STATUS[11]	(WZ) + 1 → PC
PC OUT STATUS[6]	PC = PC + 1 B3 → W									WZ OUT STATUS[11,12]	(WZ) + 1 → PC
PC OUT STATUS[6]	PC = PC + 1 B3 → W		SP OUT STATUS[16]	(PCH) → DATA BUS SP = SP - 1		SP OUT STATUS[16]	(PCL) → DATA BUS			WZ OUT STATUS[11]	(WZ) + 1 → PC
PC OUT STATUS[6]	PC = PC + 1 B3 → W[13]		SP OUT STATUS[16]	(PCH) → DATA BUS SP = SP - 1		SP OUT STATUS[16]	(PCL) → DATA BUS			WZ OUT STATUS[11,12]	(WZ) + 1 → PC
SP OUT STATUS[15]	SP = SP + 1 DATA → W									WZ OUT STATUS[11]	(WZ) + 1 → PC
SP OUT STATUS[15]	SP = SP + 1 DATA → W									WZ OUT STATUS[11,12]	(WZ) + 1 → PC
SP OUT STATUS[16]	(TMP = 00NNN000) → Z (PCL) → DATA BUS									WZ OUT STATUS[11]	(WZ) + 1 → PC
SP OUT STATUS[16]	(H) → DATA BUS										
SP OUT STATUS[16]	FLAGS → DATA BUS										
SP OUT STATUS[15]	SP = SP + 1 DATA → rh										
SP OUT STATUS[15]	SP = SP + 1 DATA → A										
SP OUT STATUS[15]	DATA → W		SP OUT STATUS[16]	(H) → DATA BUS		SP OUT STATUS[16]	(L) → DATA BUS		(WZ) → HL		
WZ OUT STATUS[18]	DATA → A										
WZ OUT STATUS[18]	(A) → DATA BUS										

**NOTES:**

1. The first memory cycle (M1) is always an instruction fetch; the first (or only) byte, containing the op code, is fetched during this cycle.
2. If the READY input from memory is not high during T2 of each memory cycle, the processor will enter a wait state (TW) until READY is sampled as high.
3. States T4 and T5 are present, as required, for operations which are completely internal to the CPU. The contents of the internal bus during T4 and T5 are available at the data bus; this is designed for testing purposes only. An "X" denotes that the state is present, but is only used for such internal operations as instruction decoding.
4. Only register pairs rp = B (registers B and C) or rp = D (registers D and E) may be specified.
5. These states are skipped.
6. Memory read sub-cycles; an instruction or data word will be read.
7. Memory write sub-cycle.
8. The READY signal is not required during the second and third sub-cycles (M2 and M3). The HOLD signal is accepted during M2 and M3. The SYNC signal is not generated during M2 and M3. During the execution of DAD, M2 and M3 are required for an internal register-pair add; memory is not referenced.
9. The results of these arithmetic, logical or rotate instructions are not moved into the accumulator (A) until state T2 of the next instruction cycle. That is, A is loaded while the next instruction is being fetched; this overlapping of operations allows for faster processing.
10. If the value of the least significant 4-bits of the accumulator is greater than 9 or if the auxiliary carry bit is set, 6 is added to the accumulator. If the value of the most significant 4-bits of the accumulator is now greater than 9, or if the carry bit is set, 6 is added to the most significant 4-bits of the accumulator.
11. This represents the first sub-cycle (the instruction fetch) of the next instruction cycle.

12. If the condition was met, the contents of the register pair WZ are output on the address lines (A<sub>0-15</sub>) instead of the contents of the program counter (PC).

13. If the condition was not met, sub-cycles M4 and M5 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.

14. If the condition was not met, sub-cycles M2 and M3 are skipped; the processor instead proceeds immediately to the instruction fetch (M1) of the next instruction cycle.

15. Stack read sub-cycle.

16. Stack write sub-cycle.

17. CONDITION	CCC
NZ — not zero (Z = 0)	000
Z — zero (Z = 1)	001
NC — no carry (CY = 0)	010
C — carry (CY = 1)	011
PO — parity odd (P = 0)	100
PE — parity even (P = 1)	101
P — plus (S = 0)	110
M — minus (S = 1)	111

18. I/O sub-cycle: the I/O port's 8-bit select code is duplicated on address lines 0-7 (A<sub>0-7</sub>) and 8-15 (A<sub>8-15</sub>).

19. Output sub-cycle.

20. The processor will remain idle in the halt state until an interrupt, a reset or a hold is accepted. When a hold request is accepted, the CPU enters the hold mode; after the hold mode is terminated, the processor returns to the halt state. After a reset is accepted, the processor begins execution at memory location zero. After an interrupt is accepted, the processor executes the instruction forced onto the data bus (usually a restart instruction).

SSS or DDD	Value	rp	Value
A	111	B	00
B	000	D	01
C	001	H	10
D	010	SP	11
E	011		
H	100		
L	101		

# CHAPTER 3 INTERFACING THE 8080

This chapter will illustrate, in detail, how to interface the 8080 CPU with Memory and I/O. It will also show the benefits and tradeoffs encountered when using a variety of system architectures to achieve higher throughput, decreased component count or minimization of memory size.

8080 Microcomputer system design lends itself to a simple, modular approach. Such an approach will yield the designer a reliable, high performance system that contains a minimum component count and is easy to manufacture and maintain.

The overall system can be thought of as a simple block diagram. The three (3) blocks in the diagram represent the functions common to any computer system.

**CPU Module\*** Contains the Central Processing Unit, system timing and interface circuitry to Memory and I/O devices.

**Memory** Contains Read Only Memory (ROM) and Read/Write Memory (RAM) for program and data storage.

**I/O** Contains circuitry that allows the computer system to communicate with devices or structures existing outside of the CPU or Memory array.

for example: Keyboards, Floppy Disks, Paper Tape, etc.

There are three busses that interconnect these blocks:

**Data Bus†** A bi-directional path on which data can flow between the CPU and Memory or I/O.

**Address Bus** A uni-directional group of lines that identify a particular Memory location or I/O device.

\*"Module" refers to a functional block, it does not reference a printed circuit board manufactured by INTEL.

†"Bus" refers to a set of signals grouped together because of the similarity of their functions.

**Control Bus** A uni-directional set of signals that indicate the type of activity in current process.

- Type of activities:
1. Memory Read
  2. Memory Write
  3. I/O Read
  4. I/O Write
  5. Interrupt Acknowledge

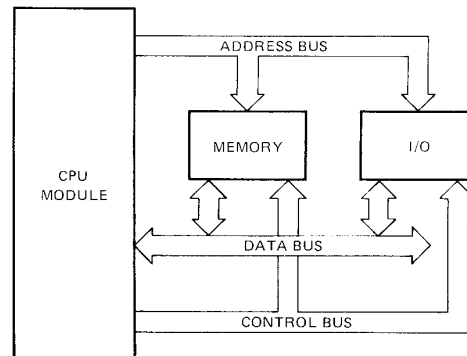


Figure 3-1. Typical Computer System Block Diagram

## Basic System Operation

1. The CPU Module issues an activity command on the Control Bus.
2. The CPU Module issues a binary code on the Address Bus to identify which particular Memory location or I/O device will be involved in the current process activity.
3. The CPU Module receives or transmits data with the selected Memory location or I/O device.
4. The CPU Module returns to ① and issues the next activity command.

It is easy to see at this point that the CPU module is the central element in any computer system.

The following pages will cover the detailed design of the CPU Module with the 8080. The three Busses (Data, Address and Control) will be developed and the interconnection to Memory and I/O will be shown.

Design philosophies and system architectures presented in this manual are consistent with product development programs underway at INTEL for the MCS-80. Thus, the designer who uses this manual as a guide for his total system engineering is assured that all new developments in components and software for MCS-80 from INTEL will be compatible with his design approach.

### CPU Module Design

The CPU Module contains three major areas:

1. The 8080 Central Processing Unit
2. A Clock Generator and High Level Driver
3. A bi-directional Data Bus Driver and System Control Logic

The following will discuss the design of the three major areas contained in the CPU Module. This design is presented as an alternative to the Intel® 8224 Clock Generator and Intel 8228 System Controller. By studying the alternative approach, the designer can more clearly see the considerations involved in the specification and engineering of the 8224 and 8228. Standard TTL components and Intel general purpose peripheral devices are used to implement

the design and to achieve operational characteristics that are as close as possible to those of the 8224 and 8228. Many auxiliary timing functions and features of the 8224 and 8228 are too complex to practically implement in standard components, so only the basic functions of the 8224 and 8228 are generated. Since significant benefits in system timing and component count reduction can be realized by using the 8224 and 8228, this is the preferred method of implementation.

#### 1. 8080 CPU

The operation of the 8080 CPU was covered in previous chapters of this manual, so little reference will be made to it in the design of the Module.

#### 2. Clock Generator and High Level Driver

The 8080 is a dynamic device, meaning that its internal storage elements and logic circuitry require a timing reference (Clock), supplied by external circuitry, to refresh and provide timing control signals.

The 8080 requires two (2) such Clocks. Their waveforms must be non-overlapping, and comply with the timing and levels specified in the 8080 A.C. and D.C. Characteristics, page 5-15.

#### Clock Generator Design

The Clock Generator consists of a crystal controlled,

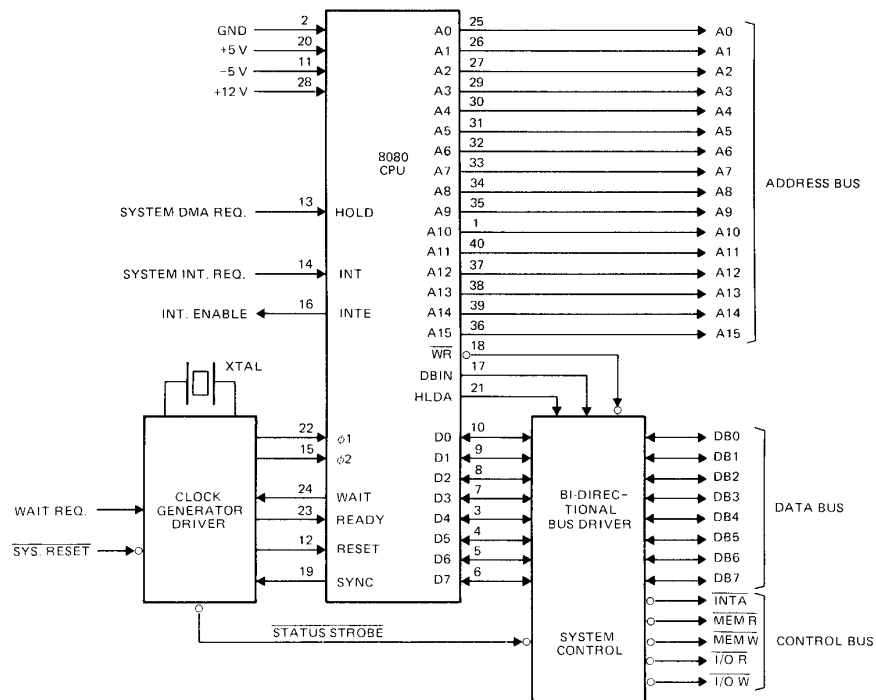


Figure 3-2. 8080 CPU Interface

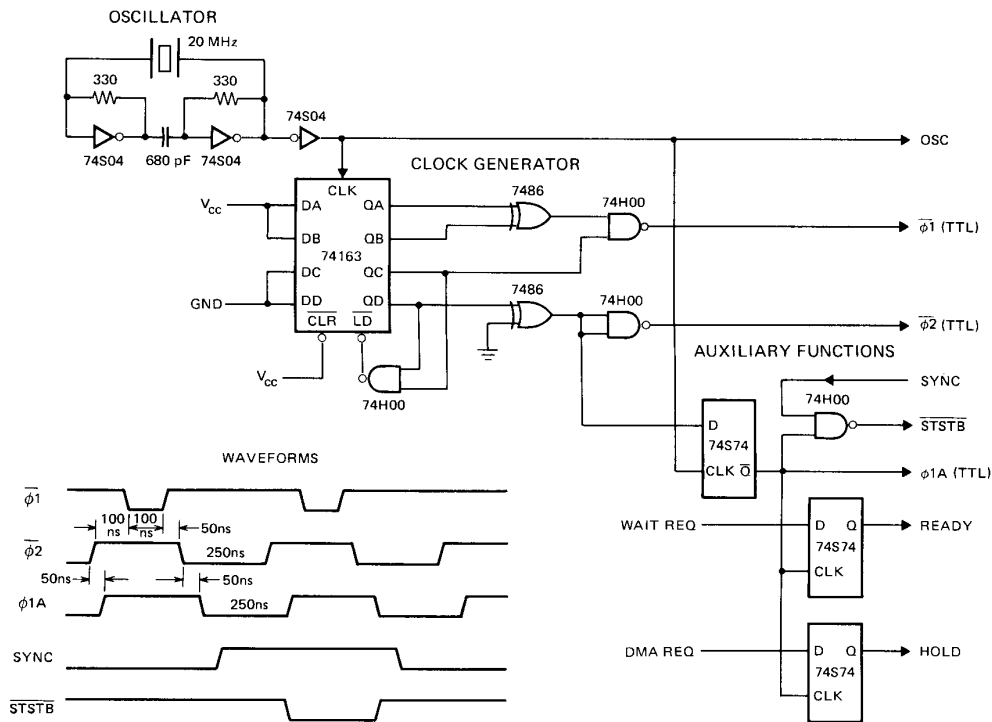


Figure 3-3. 8080 Clock Generator

20 MHz oscillator, a four bit counter, and gating circuits.

The oscillator provides a 20 MHz signal to the input of a four (4) bit, presetable, synchronous, binary counter. By presetting the counter as shown in figure 3-3 and clocking it with the 20 MHz signal, a simple decoding of the counters outputs using standard TTL gates, provides proper timing for the two (2) 8080 clock inputs.

Note that the timing must actually be measured at the output of the High Level Driver to take into account the added delays and waveform distortions within such a device.

#### High Level Driver Design

The voltage level of the clocks for the 8080 is not TTL compatible like the other signals that input to the 8080. The voltage swing is from .6 volts ( $V_{ILC}$ ) to 11 volts ( $V_{IHC}$ ) with risetimes and falltimes under 50 ns. The Capacitive Drive is 20 pf (max.). Thus, a High Level Driver is required to interface the outputs of the Clock Generator (TTL) to the 8080.

The two (2) outputs of the Clock Generator are capacitively coupled to a dual- High Level clock driver. The driver must be capable of complying with the 8080 clock input specifications, page 5-15. A driver of this type usually has little problem supplying the

positive transition when biased from the 8080  $V_{DD}$  supply (12V) but to achieve the low voltage specification ( $V_{ILC}$ ) .8 volts Max. the driver is biased to the 8080  $V_{BB}$  supply (-5V). This allows the driver to swing from GND to  $V_{DD}$  with the aid of a simple resistor divider.

A low resistance series network is added between the driver and the 8080 to eliminate any overshoot of the pulsed waveforms. Now a circuit is apparent that can easily comply with the 8080 specifications. In fact rise and falltimes of this design are typically less than 10 ns.

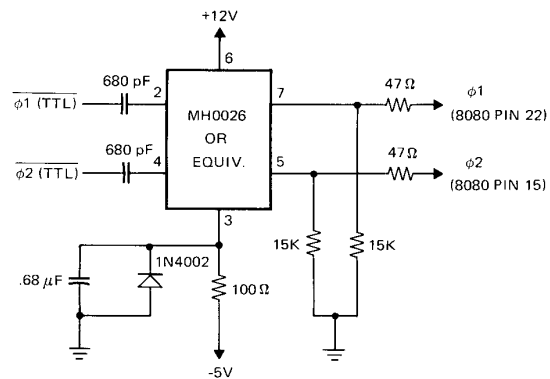


Figure 3-4. High Level Driver

### Auxiliary Timing Signals and Functions

The Clock Generator can also be used to provide other signals that the designer can use to simplify large system timing or the interface to dynamic memories.

Functions such as power-on reset, synchronization of external requests (HOLD, READY, etc.) and single step, could easily be added to the Clock Generator to further enhance its capabilities.

For instance, the 20 MHz signal from the oscillator can be buffered so that it could provide the basis for communication baud rate generation.

The Clock Generator diagram also shows how to generate an advanced timing signal ( $\phi 1A$ ) that is handy to use in clocking "D" type flipflops to synchronize external requests. It can also be used to generate a strobe ( $\overline{STSTB}$ ) that is the latching signal for the status information which is available on the Data Bus at the beginning of each machine cycle. A simple gating of the SYNC signal from the 8080 and the advanced ( $\phi 1A$ ) will do the job. See Figure 3-3.

### 3. Bi-Directional Bus Driver and System Control Logic

The system Memory and I/O devices communicate with the CPU over the bi-directional Data Bus. The system Control Bus is used to gate data on and off the Data Bus within the proper timing sequences as dictated by the operation of the 8080 CPU. The data lines of the 8080 CPU, Memory and I/O devices are 3-state in nature, that is, their output drivers have the ability to be forced into a high-impedance mode and are, effectively, removed from the circuit. This 3-state bus technique allows the designer to construct a system around a single, eight (8) bit parallel, bi-directional Data Bus and simply gate the information on or off this bus by selecting or deselecting (3-stating) Memory and I/O devices with signals from the Control Bus.

#### Bi-Directional Data Bus Driver Design

The 8080 Data Bus (D7-D0) has two (2) major areas of concern for the designer:

1. Input Voltage level ( $V_{IH}$ ) 3.3 volts minimum.
2. Output Drive Capability ( $I_{OL}$ ) 1.7 mA maximum.

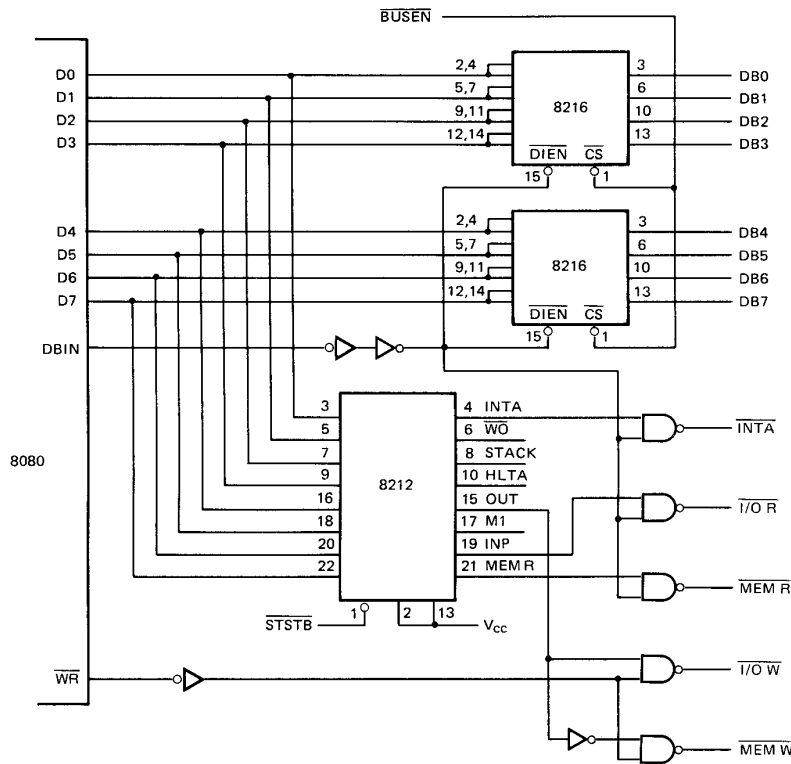


Figure 3-5. 8080 System Control



The input level specification implies that any semiconductor memory or I/O device connected to the 8080 Data Bus must be able to provide a minimum of 3.3 volts in its high state. Most semiconductor memories and standard TTL I/O devices have an output capability of between 2.0 and 2.8 volts, obviously a direct connection onto the 8080 Data Bus would require pullup resistors, whose value should not affect the bus speed or stress the drive capability of the memory or I/O components.

The 8080A output drive capability ( $I_{OL}$ ) 1.9mA max. is sufficient for small systems where Memory size and I/O requirements are minimal and the entire system is contained on a single printed circuit board. Most systems however, take advantage of the high-performance computing power of the 8080 CPU and thus a more typical system would require some form of buffering on the 8080 Data Bus to support a larger array of Memory and I/O devices which are likely to be on separate boards.

A device specifically designed to do this buffering function is the INTEL<sup>®</sup> 8216, a (4) four bit bi-directional bus driver whose input voltage level is compatible with standard TTL devices and semiconductor memory components, and has output drive capability of 50 mA. At the 8080 side, the 8216 has a "high" output of 3.65 volts that not only meets the 8080 input spec but provides the designer with a worse case 350 mV noise margin.

A pair of 8216's are connected directly to the 8080 Data Bus (D7-D0) as shown in figure 3-5. Note that the DBIN signal from the 8080 is connected to the direction control input ( $\overline{DIEN}$ ) so the correct flow of data on the bus is maintained. The chip select ( $\overline{CS}$ ) of the 8216 is connected to BUS ENABLE ( $\overline{BUSEN}$ ) to allow for DMA activities by deselecting the Data Bus Buffer and forcing the outputs of the 8216's into their high impedance (3-state) mode. This allows other devices to gain access to the data bus (DMA).

#### System Control Logic Design

The Control Bus maintains discipline of the bi-directional Data Bus, that is, it determines what type of device will have access to the bus (Memory or I/O) and generates signals to assure that these devices transfer Data with the 8080 CPU within the proper timing "windows" as dictated by the CPU operational characteristics.

As described previously, the 8080 issues Status information at the beginning of each Machine Cycle on its Data Bus to indicate what operation will take place during that cycle. A simple (8) bit latch, like an INTEL<sup>®</sup> 8212, connected directly to the 8080 Data Bus (D7-D0) as shown in figure 3-5 will store the

Status information. The signal that loads the data into the Status Latch comes from the Clock Generator, it is Status Strobe ( $\overline{STSTB}$ ) and occurs at the start of each Machine Cycle.

Note that the Status Latch is connected onto the 8080 Data Bus (D7-D0) before the Bus Buffer. This is to maintain the integrity of the Data Bus and simplify Control Bus timing in DMA dependent environments.

As shown in the diagram, a simple gating of the outputs of the Status Latch with the DBIN and  $\overline{WR}$  signals from the 8080 generate the (4) four Control signals that make up the basic Control Bus.

These four signals: 1. Memory Read ( $\overline{MEM R}$ )

2. Memory Write ( $\overline{MEM W}$ )

3. I/O Read ( $\overline{I/O R}$ )

4. I/O Write ( $\overline{I/O W}$ )

connect directly to the MCS-80 component "family" of ROMs, RAMs and I/O devices.

A fifth signal, Interrupt Acknowledge ( $\overline{INTA}$ ) is added to the Control Bus by gating data off the Status Latch with the DBIN signal from the 8080 CPU. This signal is used to enable the Interrupt Instruction Port which holds the RST instruction onto the Data Bus.

Other signals that are part of the Control Bus such as  $\overline{WO}$ , Stack and M1 are present to aid in the testing of the System and also to simplify interfacing the CPU to dynamic memories or very large systems that require several levels of bus buffering.

#### Address Buffer Design

The Address Bus (A15-A0) of the 8080, like the Data Bus, is sufficient to support a small system that has a moderate size Memory and I/O structure, confined to a single card. To expand the size of the system that the Address Bus can support a simple buffer can be added, as shown in figure 3-6. The INTEL<sup>®</sup> 8212 or 8216 is an excellent device for this function. They provide low input loading (.25 mA), high output drive and insert a minimal delay in the System Timing.

Note that BUS ENABLE ( $\overline{BUSEN}$ ) is connected to the buffers so that they are forced into their high-impedance (3-state) mode during DMA activities so that other devices can gain access to the Address Bus.

## INTERFACING THE 8080 CPU TO MEMORY AND I/O DEVICES

The 8080 interfaces with standard semiconductor Memory components and I/O devices. In the previous text the proper control signals and buffering were developed which will produce a simple bus system similar to the basic system example shown at the beginning of this chapter.

In Figure 3-6 a simple, but exact 8080 typical system is shown that can be used as a guide for any 8080 system, regardless of size or complexity. It is a "three bus" architecture, using the signals developed in the CPU module.

Note that Memory and I/O devices interface in the same manner and that their isolation is only a function of the definition of the Read-Write signals on the Control Bus. This allows the 8080 system to be configured so that Memory and I/O are treated as a single array (memory mapped I/O) for small systems that require high thruput and have less than 32K memory size. This approach will be brought out later in the chapter.

### ROM INTERFACE

A ROM is a device that stores data in the form of Program or other information such as "look-up tables" and is only read from, thus the term Read Only Memory. This type of memory is generally non-volatile, meaning that when the power is removed the information is retained.

This feature eliminates the need for extra equipment like tape readers and disks to load programs initially, an important aspect in small system design.

Interfacing standard ROMs, such as the devices shown in the diagram is simple and direct. The output Data lines are connected to the bi-directional Data Bus, the Address inputs tie to the Address bus with possible decoding of the most significant bits as "chip selects" and the  $\overline{\text{MEMR}}$  signal from the Control Bus connected to a "chip select" or data buffer. Basically, the CPU issues an address during the first portion of an instruction or data fetch (T1 & T2). This value on the Address Bus selects a specific location within the ROM, then depending on the ROM's delay (access time) the data stored at the addressed location is present at the Data output lines. At this time (T3) the CPU Data Bus is in the "input Mode" and the control logic issues a Memory Read command ( $\overline{\text{MEMR}}$ ) that gates the addressed data on to the Data Bus.

### RAM INTERFACE

A RAM is a device that stores data. This data can be program, active "look-up tables," temporary values or external stacks. The difference between RAM and ROM is that data can be written into such devices and are in essence, Read/Write storage elements. RAMs do not hold their data when power is removed so in the case where Program or "look-up tables" data is stored a method to load

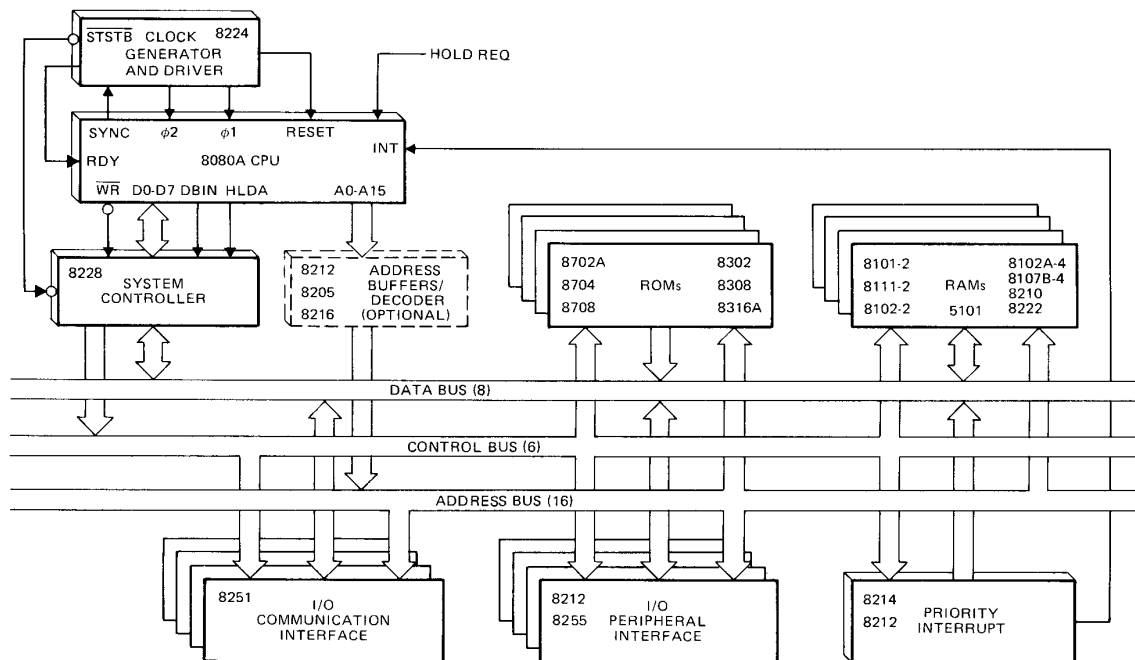


Figure 3-6. Microcomputer System

RAM memory must be provided, such as: Floppy Disk, Paper Tape, etc.

The CPU treats RAM in exactly the same manner as ROM for addressing data to be read. Writing data is very similar; the RAM is issued an address during the first portion of the Memory Write cycle (T1 & T2) in T3 when the data that is to be written is output by the CPU and is stable on the bus an  $\overline{\text{MEMW}}$  command is generated. The  $\overline{\text{MEMW}}$  signal is connected to the R/W input of the RAM and strobes the data into the addressed location.

In Figure 3-7 a typical Memory system is illustrated to show how standard semiconductor components interface to the 8080 bus. The memory array shown has 8K bytes (8 bits/byte) of ROM storage, using four Intel<sup>®</sup> 8216As and 512 bytes of RAM storage, using Intel 8111 static RAMs. The basic interface to the bus structure detailed here is common to almost any size memory. The only addition that might have to be made for larger systems is more buffers (8216/8212) and decoders (8205) for generating "chip selects."

The memories chosen for this example have an access time of 850 nS (max) to illustrate that slower, economical devices can be easily interfaced to the 8080 with little effect on performance. When the 8080 is operated from a clock generator with a tCY of 500 nS the required memory access time is Approx. 450-550 nS. See detailed timing specification Pg. 5-16. Using memory devices of this speed such as Intel<sup>®</sup> 8308, 8102A, 8107A, etc. the READY input to the 8080 CPU can remain "high" because no "wait" states are required. Note that the bus interface to memory shown in Figure 3-7 remains the same. However, if slower memories are to be used, such as the devices illustrated (8316A, 8111) that have access times slower than the minimum requirement a simple logic control of the READY input to the 8080 CPU will insert an extra "wait state" that is equal to one or more clock periods as an access time "adjustment" delay to compensate. The effect of the extra "wait" state is naturally a slower execution time for the instruction. A single "wait" changes the basic instruction cycle to 2.5 microSeconds.

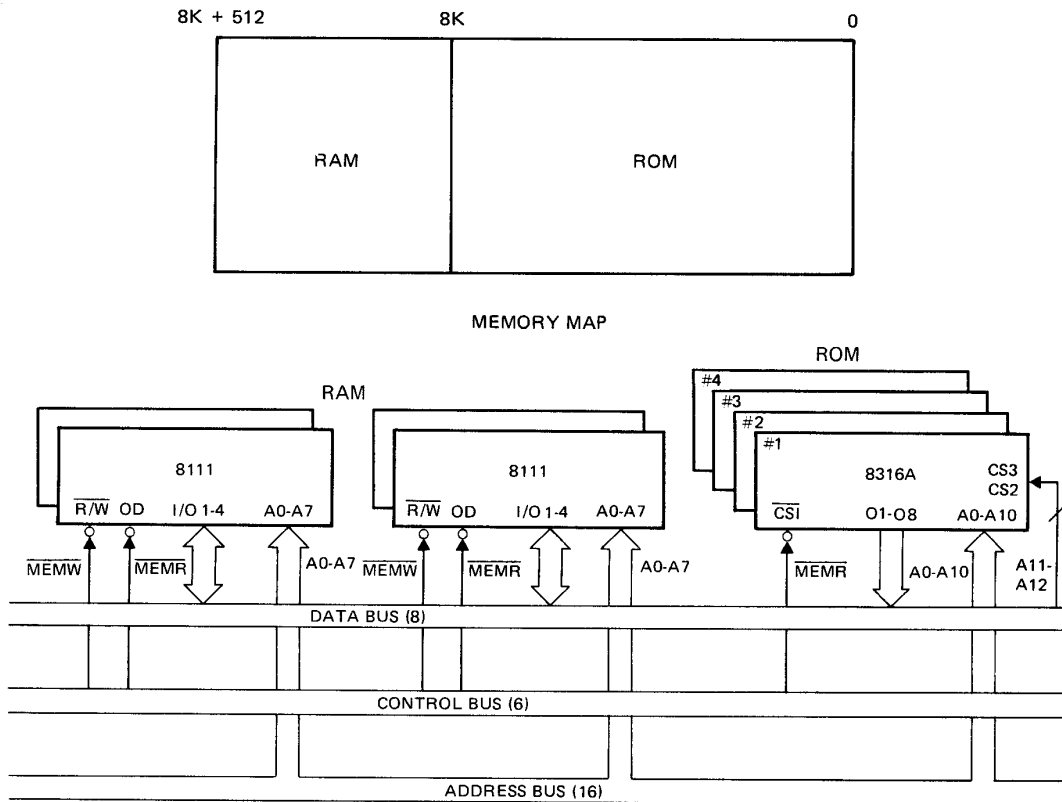


Figure 3-7. Typical Memory Interface

## I/O INTERFACE

### General Theory

As in any computer based system, the 8080 CPU must be able to communicate with devices or structures that exist outside its normal memory array. Devices like keyboards, paper tape, floppy disks, printers, displays and other control structures are used to input information into the 8080 CPU and display or store the results of the computational activity.

Probably the most important and strongest feature of the 8080 Microcomputer System is the flexibility and power of its I/O structure and the components that support it. There are many ways to structure the I/O array so that it will "fit" the total system environment to maximize efficiency and minimize component count.

The basic operation of the I/O structure can best be viewed as an array of single byte memory locations that can be Read from or Written into. The 8080 CPU has special instructions devoted to managing such transfers (IN, OUT). These instructions generally isolate memory and I/O arrays so that memory address space is not effected by the I/O structure and the general concept is that of a simple transfer to or from the Accumulator with an addressed "PORT". Another method of I/O architecture is to treat the I/O structure as part of the Memory array. This is generally referred to as "Memory Mapped I/O" and provides the designer with a powerful new "instruction set" devoted to I/O manipulation.

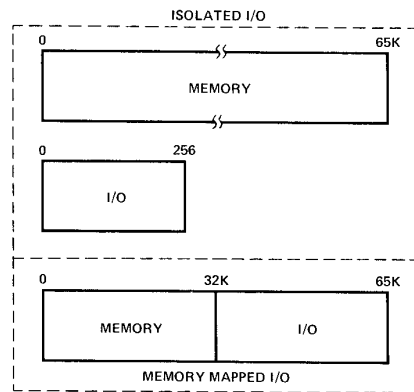


Figure 3-8. Memory/I/O Mapping.

### Isolated I/O

In Figure 3-9 the system control signals, previously detailed in this chapter, are shown. This type of I/O architecture separates the memory address space from the I/O address space and uses a conceptually simple transfer to or from Accumulator technique. Such an architecture is easy to understand because I/O communicates only with the Accumulator using the IN or OUT instructions. Also because of the isolation of memory and I/O, the full address space (65K) is unaffected by I/O addressing.

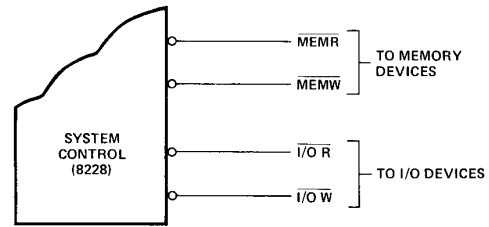


Figure 3-9. Isolated I/O.

### Memory Mapped I/O

By assigning an area of memory address space as I/O a powerful architecture can be developed that can manipulate I/O using the same instructions that are used to manipulate memory locations. Thus, a "new" instruction set is created that is devoted to I/O handling.

As shown in Figure 3-10, new control signals are generated by gating the MEMR and MEMW signals with A<sub>15</sub>, the most significant address bit. The new I/O control signals connect in exactly the same manner as Isolated I/O, thus the system bus characteristics are unchanged.

By assigning A<sub>15</sub> as the I/O "flag", a simple method of I/O discipline is maintained:

- If A<sub>15</sub> is a "zero" then Memory is active.
- If A<sub>15</sub> is a "one" then I/O is active.

Other address bits can also be used for this function. A<sub>15</sub> was chosen because it is the most significant address bit so it is easier to control with software and because it still allows memory addressing of 32K.

I/O devices are still considered addressed "ports" but instead of the Accumulator as the only transfer medium any of the internal registers can be used. All instructions that could be used to operate on memory locations can be used in I/O.

Examples:

MOV r, M	(Input Port to any Register)
MOV M, r	(Output any Register to Port)
MVI M	(Output immediate data to Port)
LDA	(Input to ACC)
STA	(Output from ACC to Port)
LHLD	(16 Bit Input)
SHLD	(16 Bit Output)
ADD M	(Add Port to ACC)
ANA M	("AND" Port with ACC)

It is easy to see that from the list of possible "new" instructions that this type of I/O architecture could have a drastic effect on increased system throughput. It is conceptually more difficult to understand than Isolated I/O and it does limit memory address space, but Memory Mapped I/O can mean a significant increase in overall speed and at the same time reducing required program memory area.

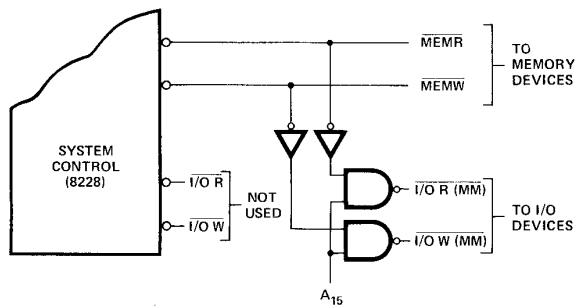


Figure 3-10. Memory Mapped I/O.

### I/O Addressing

With both systems of I/O structure the addressing of each device can be configured to optimize efficiency and reduce component count. One method, the most common, is to decode the address bus into exclusive "chip selects" that enable the addressed I/O device, similar to generating chip-selects in memory arrays.

Another method is called "linear select". In this method, instead of decoding the Address Bus, a singular bit from the bus is assigned as the exclusive enable for a specific I/O device. This method, of course, limits the number of I/O devices that can be addressed but eliminates the need for extra decoders, an important consideration in small system design.

A simple example illustrates the power of such a flexible I/O structure. The first example illustrates the format of the second byte of the IN or OUT instruction using the Isolated I/O technique. The devices used are Intel®8255 Programmable Peripheral Interface units and are linear selected. Each device has three ports and from the format it can be seen that six devices can be addressed without additional decoders.

### EXAMPLE #1

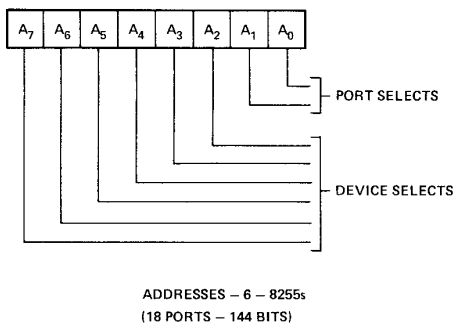


Figure 3-11. Isolated I/O - (Linear Select) (8255)

The second example uses Memory Mapped I/O and linear select to show how thirteen devices (8255) can be addressed without the use of extra decoders. The format shown could be the second and third bytes of the LDA or STA instructions or any other instructions used to manipulate I/O using the Memory Mapped technique.

It is easy to see that such a flexible I/O structure, that can be "tailored" to the overall system environment, provides the designer with a powerful tool to optimize efficiency and minimize component count.

### EXAMPLE #2

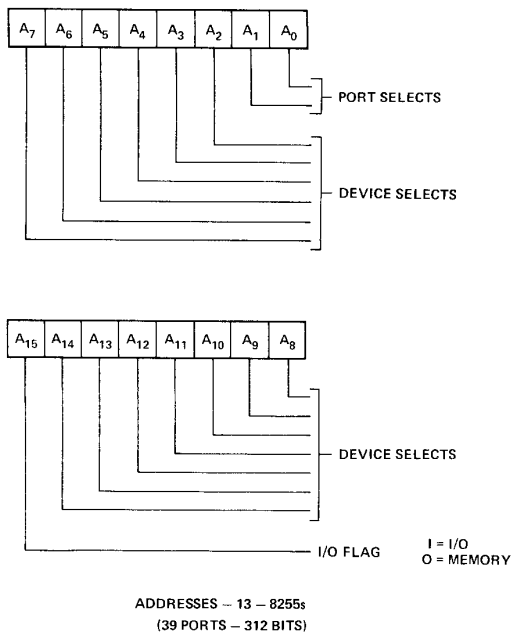


Figure 3-12. Memory Mapped I/O - (Linear Select (8255))

### I/O Interface Example

In Figure 3-16 a typical I/O system is shown that uses a variety of devices (8212, 8251 and 8255). It could be used to interface the peripherals around an intelligent CRT terminals; keyboards, display, and communication interface. Another application could be in a process controller to interface sensors, relays, and motor controls. The limitation of the application area for such a circuit is solely that of the designers imagination.

The I/O structure shown interfaces to the 8080 CPU using the bus architecture developed previously in this chapter. Either Isolated or Memory Mapped techniques can be used, depending on the system I/O environment.

The 8251 provides a serial data communication interface so that the system can transmit and receive data over communication links such as telephone lines.

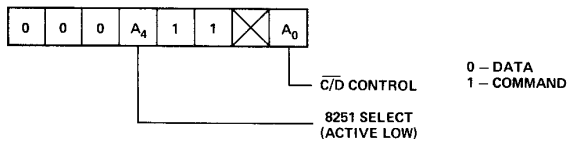


Figure 3-13. 8251 Format.

The two (2) 8255s provide twenty four bits each of programmable I/O data and control so that keyboards, sensors, paper tape, etc., can be interfaced to the system.

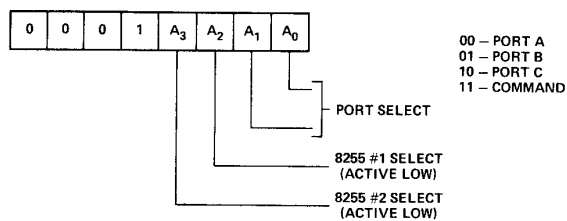


Figure 3-14. 8255 Format.

The three 8212s can be used to drive long lines or LED indicators due to their high drive capability. (15mA)

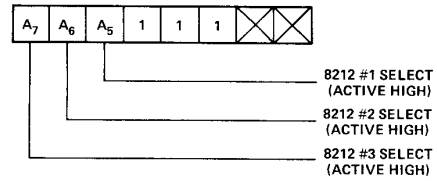


Figure 3-15. 8212 Format.

Addressing the structure is described in the formats illustrated in Figures 3-13, 3-14, 3-15. Linear Select is used so that no decoders are required thus, each device has an exclusive "enable bit".

The example shows how a powerful yet flexible I/O structure can be created using a minimum component count with devices that are all members of the 8080 Microcomputer System.

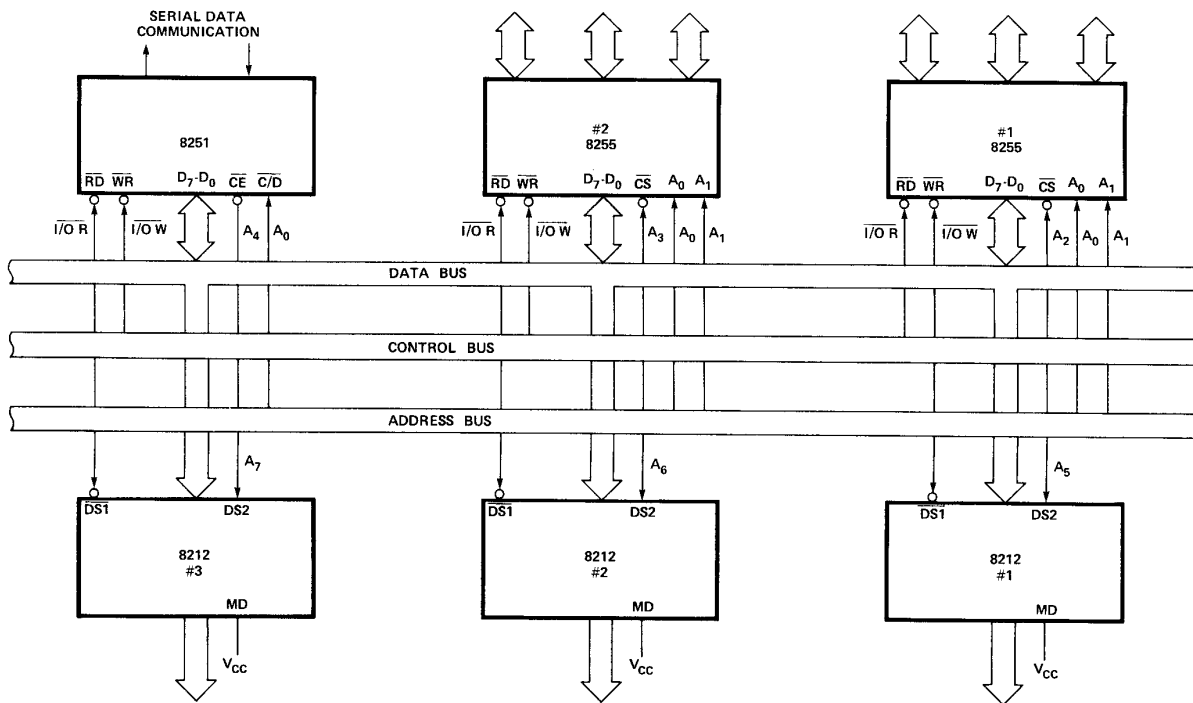


Figure 3-16. Typical I/O Interface.

## CHAPTER 4 INSTRUCTION SET

A computer, no matter how sophisticated, can only do what it is “told” to do. One “tells” the computer what to do via a series of coded instructions referred to as a **Program**. The realm of the programmer is referred to as **Software**, in contrast to the **Hardware** that comprises the actual computer equipment. A computer’s software refers to all of the programs that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer’s **Instruction Set**.

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logical operations (e.g., OR the contents of two registers) and register operate instructions (e.g., increment a register) are included in the instruction set. A computer’s instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide **Conditional Instructions**. A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a program with a decision-making capability.

By logically organizing a sequence of instructions into a coherent program, the programmer can “tell” the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1’s and 0’s), that is called **Machine Code**. Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There

are programs available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is **Assembly Language**. A unique assembly language mnemonic is assigned to each of the computer’s instructions. The programmer can write a program (called the **Source Program**) using these mnemonics and certain operands; the source program is then converted into machine instructions (called the **Object Code**). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an **Assembler** program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

### THE 8080 INSTRUCTION SET

The 8080 instruction set includes five different types of instructions:

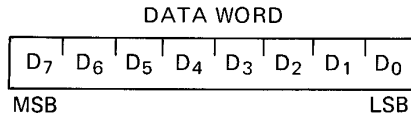
- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group** — add, subtract, increment or decrement data in registers or in memory
- **Logical Group** — AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group** — conditional and unconditional jump instructions, subroutine call instructions and return instructions
- **Stack, I/O and Machine Control Group** — includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

### Instruction and Data Formats:

Memory for the 8080 is organized into 8-bit quantities, called Bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

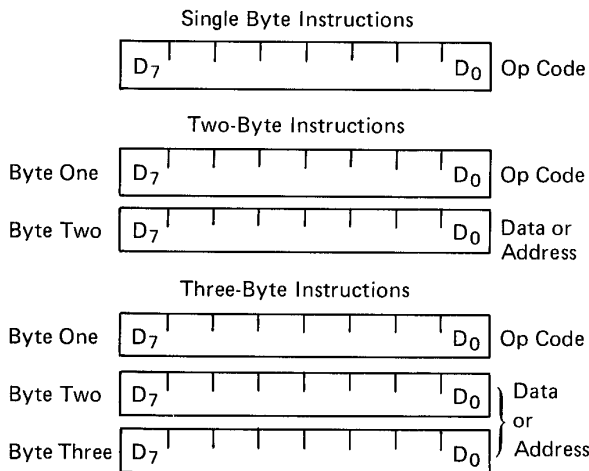
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the **Least Significant Bit (LSB)**, and BIT 7 (of an 8 bit number) is referred to as the **Most Significant Bit (MSB)**.

The 8080 program instructions may be one, two or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



### Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- **Direct** – Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- **Register** – The instruction specifies the register or register-pair in which the data is located.
- **Register Indirect** – The instruction specifies a register-pair which contains the memory

address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).

- **Immediate** – The instruction contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- **Direct** – The branch instruction contains the address of the next instruction to be executed. (Except for the 'RST' instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- **Register indirect** – The branch instruction indicates a register-pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special one-byte call instruction (usually used during interrupt sequences). RST includes a three-bit field; program control is transferred to the instruction whose address is eight times the contents of this three-bit field.

### Condition Flags:

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is "set" by forcing the bit to 1; "reset" by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner:

- |         |   |
|---------|---|
| Zero:   | If the result of an instruction has the value 0, this flag is set; otherwise it is reset.   |
| Sign:   | If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.  |
| Parity: | If the modulo 2 sum of the bits of the result of the operation is 0, (i.e., if the result has even parity), this flag is set; otherwise it is reset (i.e., if the result has odd parity). |
| Carry:  | If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.                |



Auxiliary Carry: If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

### Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r,r1,r2	One of the registers A,B,C,D,E,H,L
DDD,SSS	The bit pattern designating one of the registers A,B,C,D,E,H,L (DDD=destination, SSS=source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp One of the register pairs:  
 B represents the B,C pair with B as the high-order register and C as the low-order register;  
 D represents the D,E pair with D as the high-order register and E as the low-order register;  
 H represents the H,L pair with H as the high-order register and L as the low-order register;  
 SP represents the 16-bit stack pointer register.

RP The bit pattern designating one of the register pairs B,D,H,SP:

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh	The first (high-order) register of a designated register pair.
rl	The second (low-order) register of a designated register pair.
PC	16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8 bits respectively).
SP	16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8 bits respectively).
r <sub>m</sub>	Bit m of the register r (bits are number 7 through 0 from left to right).
Z,S,P,CY,AC	The condition flags: Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.
( )	The contents of the memory location or registers enclosed in the parentheses.
←	"Is transferred to"
∧	Logical AND
∨	Exclusive OR
∇	Inclusive OR
+	Addition
-	Two's complement subtraction
*	Multiplication
↔	"Is exchanged with"
—	The one's complement (e.g., $\overline{A}$ )
n	The restart number 0 through 7
NNN	The binary representation 000 through 111 for restart number 0 through 7 respectively.

### Description Format:

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The MAC 80 assembler format, consisting of the instruction mnemonic and operand fields, is printed in **BOLDFACE** on the left side of the first line.
2. The name of the instruction is enclosed in parenthesis on the right side of the first line.
3. The next line(s) contain a symbolic description of the operation of the instruction.
4. This is followed by a narrative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.

6. The last four lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a Conditional Jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see Page 4-2) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

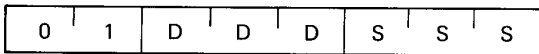
### Data Transfer Group:

This group of instructions transfers data to and from registers and memory. **Condition flags are not affected** by any instruction in this group.

#### MOV r1, r2 (Move Register)

(r1) ← (r2)

The content of register r2 is moved to register r1.

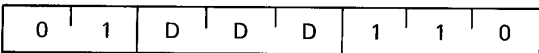


Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: none

#### MOV r, M (Move from memory)

(r) ← ((H) (L))

The content of the memory location, whose address is in registers H and L, is moved to register r.

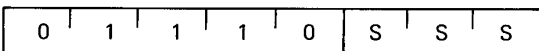


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: none

#### MOV M, r (Move to memory)

((H) (L)) ← (r)

The content of register r is moved to the memory location whose address is in registers H and L.

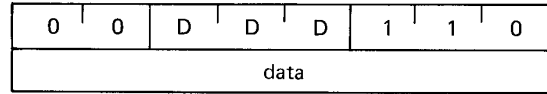


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: none

#### MVI r, data (Move Immediate)

(r) ← (byte 2)

The content of byte 2 of the instruction is moved to register r.

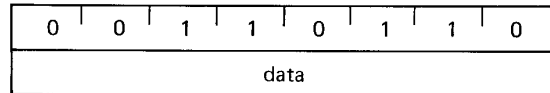


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: none

#### MVI M, data (Move to memory immediate)

((H) (L)) ← (byte 2)

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.



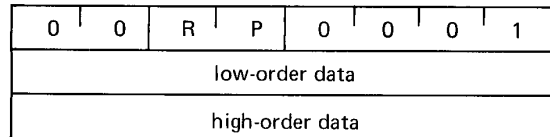
Cycles: 3  
 States: 10  
 Addressing: immed./reg. indirect  
 Flags: none

#### LXI rp, data 16 (Load register pair immediate)

(rh) ← (byte 3),

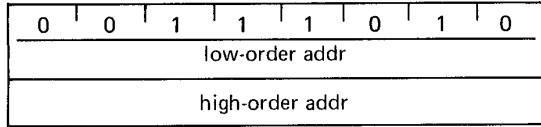
(rl) ← (byte 2)

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.



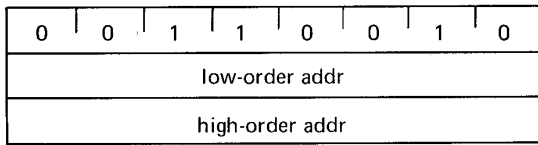
Cycles: 3  
 States: 10  
 Addressing: immediate  
 Flags: none

**LDA addr** (Load Accumulator direct)  
 $(A) \leftarrow ((\text{byte } 3)(\text{byte } 2))$   
 The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.



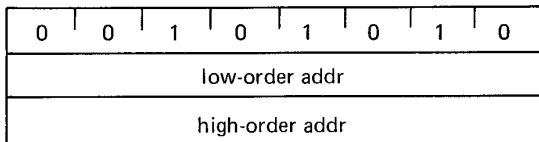
Cycles: 4  
 States: 13  
 Addressing: direct  
 Flags: none

**STA addr** (Store Accumulator direct)  
 $((\text{byte } 3)(\text{byte } 2)) \leftarrow (A)$   
 The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.



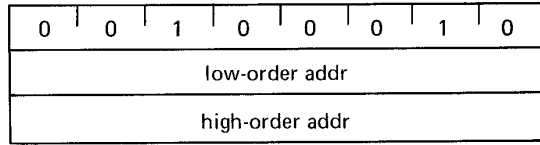
Cycles: 4  
 States: 13  
 Addressing: direct  
 Flags: none

**LHLD addr** (Load H and L direct)  
 $(L) \leftarrow ((\text{byte } 3)(\text{byte } 2))$   
 $(H) \leftarrow ((\text{byte } 3)(\text{byte } 2) + 1)$   
 The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register L. The content of the memory location at the succeeding address is moved to register H.



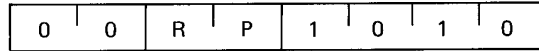
Cycles: 5  
 States: 16  
 Addressing: direct  
 Flags: none

**SHLD addr** (Store H and L direct)  
 $((\text{byte } 3)(\text{byte } 2)) \leftarrow (L)$   
 $((\text{byte } 3)(\text{byte } 2) + 1) \leftarrow (H)$   
 The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.



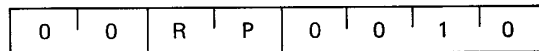
Cycles: 5  
 States: 16  
 Addressing: direct  
 Flags: none

**LDAX rp** (Load accumulator indirect)  
 $(A) \leftarrow ((rp))$   
 The content of the memory location, whose address is in the register pair rp, is moved to register A. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.



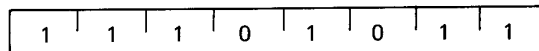
Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: none

**STAX rp** (Store accumulator indirect)  
 $((rp)) \leftarrow (A)$   
 The content of register A is moved to the memory location whose address is in the register pair rp. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.



Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: none

**XCHG** (Exchange H and L with D and E)  
 $(H) \leftrightarrow (D)$   
 $(L) \leftrightarrow (E)$   
 The contents of registers H and L are exchanged with the contents of registers D and E.



Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: none

## Arithmetic Group:

This group of instructions performs arithmetic operations on data in registers and memory.

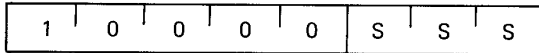
Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

### ADD r (Add Register)

$$(A) \leftarrow (A) + (r)$$

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

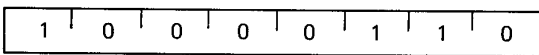


Cycles: 1  
States: 4  
Addressing: register  
Flags: Z,S,P,CY,AC

### ADD M (Add memory)

$$(A) \leftarrow (A) + ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.

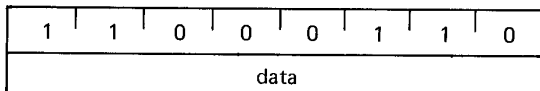


Cycles: 2  
States: 7  
Addressing: reg. indirect  
Flags: Z,S,P,CY,AC

### ADI data (Add immediate)

$$(A) \leftarrow (A) + (\text{byte 2})$$

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.

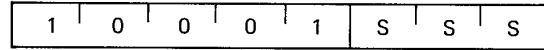


Cycles: 2  
States: 7  
Addressing: immediate  
Flags: Z,S,P,CY,AC

### ADC r (Add Register with carry)

$$(A) \leftarrow (A) + (r) + (CY)$$

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.

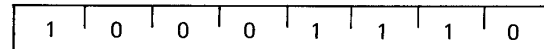


Cycles: 1  
States: 4  
Addressing: register  
Flags: Z,S,P,CY,AC

### ADC M (Add memory with carry)

$$(A) \leftarrow (A) + ((H) (L)) + (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.

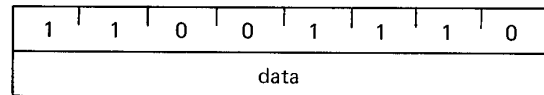


Cycles: 2  
States: 7  
Addressing: reg. indirect  
Flags: Z,S,P,CY,AC

### ACI data (Add immediate with carry)

$$(A) \leftarrow (A) + (\text{byte 2}) + (CY)$$

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.

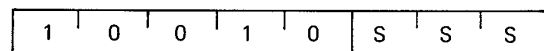


Cycles: 2  
States: 7  
Addressing: immediate  
Flags: Z,S,P,CY,AC

### SUB r (Subtract Register)

$$(A) \leftarrow (A) - (r)$$

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.

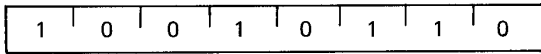


Cycles: 1  
States: 4  
Addressing: register  
Flags: Z,S,P,CY,AC

**SUB M** (Subtract memory)

$$(A) \leftarrow (A) - ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.

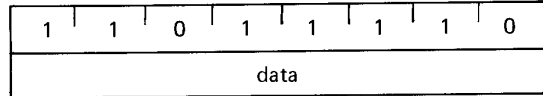


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**SBI data** (Subtract immediate with borrow)

$$(A) \leftarrow (A) - (\text{byte 2}) - (CY)$$

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

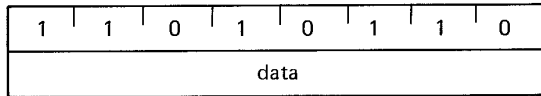


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: Z,S,P,CY,AC

**SUI data** (Subtract immediate)

$$(A) \leftarrow (A) - (\text{byte 2})$$

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.

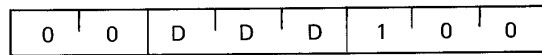


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: Z,S,P,CY,AC

**INR r** (Increment Register)

$$(r) \leftarrow (r) + 1$$

The content of register r is incremented by one. Note: All condition flags **except** CY are affected.

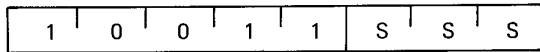


Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: Z,S,P,AC

**SBB r** (Subtract Register with borrow)

$$(A) \leftarrow (A) - (r) - (CY)$$

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

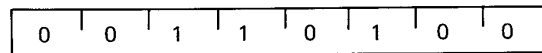


Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: Z,S,P,CY,AC

**INR M** (Increment memory)

$$((H) (L)) \leftarrow ((H) (L)) + 1$$

The content of the memory location whose address is contained in the H and L registers is incremented by one. Note: All condition flags **except** CY are affected.

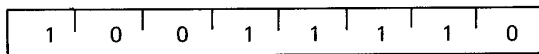


Cycles: 3  
 States: 10  
 Addressing: reg. indirect  
 Flags: Z,S,P,AC

**SBB M** (Subtract memory with borrow)

$$(A) \leftarrow (A) - ((H) (L)) - (CY)$$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

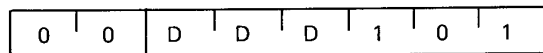


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**DCR r** (Decrement Register)

$$(r) \leftarrow (r) - 1$$

The content of register r is decremented by one. Note: All condition flags **except** CY are affected.



Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: Z,S,P,AC

**DCR M** (Decrement memory)

$$((H) (L)) \leftarrow ((H) (L)) - 1$$

The content of the memory location whose address is contained in the H and L registers is decremented by one. Note: All condition flags **except CY** are affected.

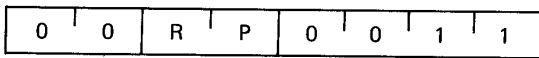


Cycles: 3  
 States: 10  
 Addressing: reg. indirect  
 Flags: Z,S,P,AC

**INX rp** (Increment register pair)

$$(rh) (rl) \leftarrow (rh) (rl) + 1$$

The content of the register pair *rp* is incremented by one. Note: **No condition flags are affected.**

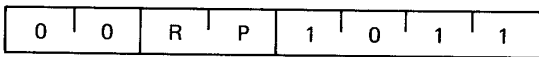


Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: none

**DCX rp** (Decrement register pair)

$$(rh) (rl) \leftarrow (rh) (rl) - 1$$

The content of the register pair *rp* is decremented by one. Note: **No condition flags are affected.**

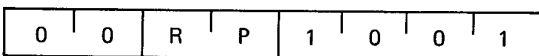


Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: none

**DAD rp** (Add register pair to H and L)

$$(H) (L) \leftarrow (H) (L) + (rh) (rl)$$

The content of the register pair *rp* is added to the content of the register pair H and L. The result is placed in the register pair H and L. Note: **Only the CY flag is affected.** It is set if there is a carry out of the double precision add; otherwise it is reset.



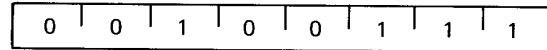
Cycles: 3  
 States: 10  
 Addressing: register  
 Flags: CY

**DAA** (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 or if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, or if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.

NOTE: All flags are affected.



Cycles: 1  
 States: 4  
 Flags: Z,S,P,CY,AC

**Logical Group:**

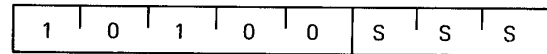
This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

**ANA r** (AND Register)

$$(A) \leftarrow (A) \wedge (r)$$

The content of register *r* is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

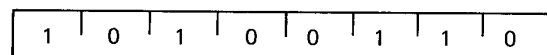


Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: Z,S,P,CY,AC

**ANA M** (AND memory)

$$(A) \leftarrow (A) \wedge ((H) (L))$$

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

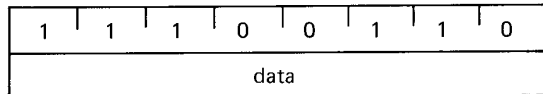


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**ANI data** (AND immediate)

$$(A) \leftarrow (A) \wedge (\text{byte } 2)$$

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

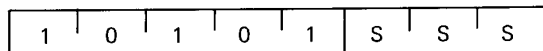


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: Z,S,P,CY,AC

**XRA r** (Exclusive OR Register)

$$(A) \leftarrow (A) \vee (r)$$

The content of register r is exclusive-or'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

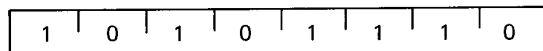


Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: Z,S,P,CY,AC

**XRA M** (Exclusive OR Memory)

$$(A) \leftarrow (A) \vee ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

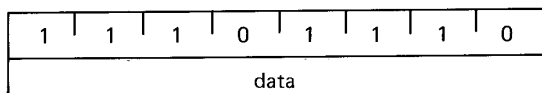


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**XRI data** (Exclusive OR immediate)

$$(A) \leftarrow (A) \vee (\text{byte } 2)$$

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

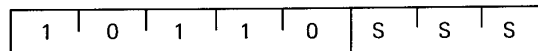


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: Z,S,P,CY,AC

**ORA r** (OR Register)

$$(A) \leftarrow (A) \vee (r)$$

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

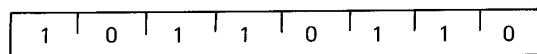


Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: Z,S,P,CY,AC

**ORA M** (OR memory)

$$(A) \leftarrow (A) \vee ((H) (L))$$

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

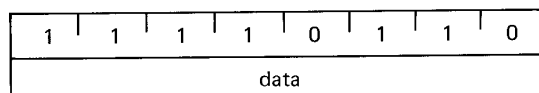


Cycles: 2  
 States: 7  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**ORI data** (OR Immediate)

$$(A) \leftarrow (A) \vee (\text{byte } 2)$$

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

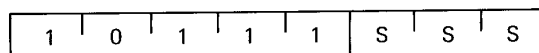


Cycles: 2  
 States: 7  
 Addressing: immediate  
 Flags: Z,S,P,CY,AC

**CMP r** (Compare Register)

$$(A) - (r)$$

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if (A) = (r). The CY flag is set to 1 if (A) < (r).**

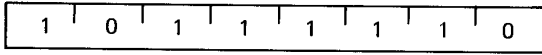


Cycles: 1  
 States: 4  
 Addressing: register  
 Flags: Z,S,P,CY,AC

**CMP M** (Compare memory)

(A) ← ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).

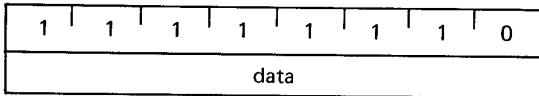


Cycles: 2  
States: 7  
Addressing: reg. indirect  
Flags: Z,S,P,CY,AC

**CPI data** (Compare immediate)

(A) ← (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).

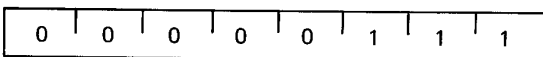


Cycles: 2  
States: 7  
Addressing: immediate  
Flags: Z,S,P,CY,AC

**RLC** (Rotate left)

(A<sub>n+1</sub>) ← (A<sub>n</sub>) ; (A<sub>0</sub>) ← (A<sub>7</sub>)  
(CY) ← (A<sub>7</sub>)

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. **Only the CY flag is affected.**

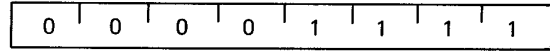


Cycles: 1  
States: 4  
Flags: CY

**RRC** (Rotate right)

(A<sub>n</sub>) ← (A<sub>n-1</sub>) ; (A<sub>7</sub>) ← (A<sub>0</sub>)  
(CY) ← (A<sub>0</sub>)

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. **Only the CY flag is affected.**

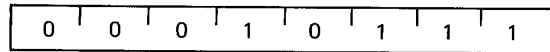


Cycles: 1  
States: 4  
Flags: CY

**RAL** (Rotate left through carry)

(A<sub>n+1</sub>) ← (A<sub>n</sub>) ; (CY) ← (A<sub>7</sub>)  
(A<sub>0</sub>) ← (CY)

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. **Only the CY flag is affected.**

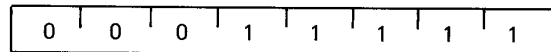


Cycles: 1  
States: 4  
Flags: CY

**RAR** (Rotate right through carry)

(A<sub>n</sub>) ← (A<sub>n+1</sub>) ; (CY) ← (A<sub>7</sub>)  
(A<sub>7</sub>) ← (CY)

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. **Only the CY flag is affected.**

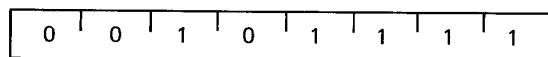


Cycles: 1  
States: 4  
Flags: CY

**CMA** (Complement accumulator)

(A) ← (A̅)

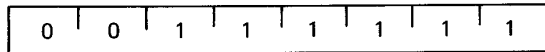
The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**



Cycles: 1  
States: 4  
Flags: none

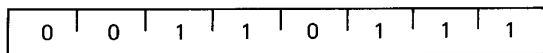


**CMC** (Complement carry)  
 (CY) ←  $\overline{\text{CY}}$   
 The CY flag is complemented. **No other flags are affected.**



Cycles: 1  
 States: 4  
 Flags: CY

**STC** (Set carry)  
 (CY) ← 1  
 The CY flag is set to 1. **No other flags are affected.**



Cycles: 1  
 States: 4  
 Flags: CY

### Branch Group:

This group of instructions alter normal sequential program flow.

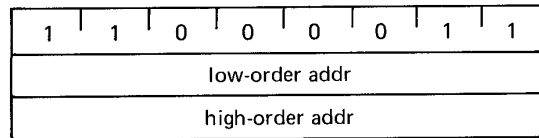
**Condition flags are not affected** by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The conditions that may be specified are as follows:

CONDITION	CCC
NZ — not zero (Z = 0)	000
Z — zero (Z = 1)	001
NC — no carry (CY = 0)	010
C — carry (CY = 1)	011
PO — parity odd (P = 0)	100
PE — parity even (P = 1)	101
P — plus (S = 0)	110
M — minus (S = 1)	111

**JMP addr** (Jump)  
 (PC) ← (byte 3) (byte 2)  
 Control is transferred to the instruction whose ad-

dress is specified in byte 3 and byte 2 of the current instruction.

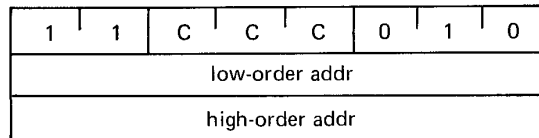


Cycles: 3  
 States: 10  
 Addressing: immediate  
 Flags: none

**Jcondition addr** (Conditional jump)

If (CCC),  
 (PC) ← (byte 3) (byte 2)

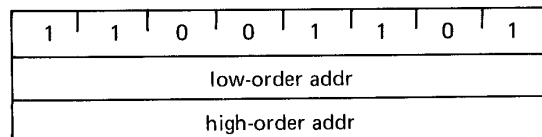
If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise, control continues sequentially.



Cycles: 3  
 States: 10  
 Addressing: immediate  
 Flags: none

**CALL addr** (Call)  
 ((SP) - 1) ← (PCH)  
 ((SP) - 2) ← (PCL)  
 (SP) ← (SP) - 2  
 (PC) ← (byte 3) (byte 2)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

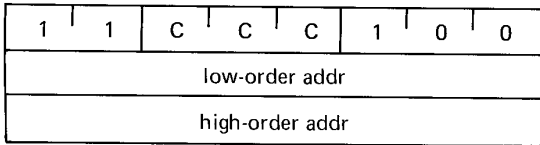


Cycles: 5  
 States: 17  
 Addressing: immediate/reg. indirect  
 Flags: none

**Ccondition addr** (Condition call)

If (CCC),  
 $((SP) - 1) \leftarrow (PCH)$   
 $((SP) - 2) \leftarrow (PCL)$   
 $(SP) \leftarrow (SP) - 2$   
 $(PC) \leftarrow (\text{byte 3}) (\text{byte 2})$

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.

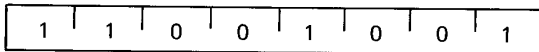


Cycles: 3/5  
 States: 11/17  
 Addressing: immediate/reg. indirect  
 Flags: none

**RET** (Return)

$(PCL) \leftarrow ((SP));$   
 $(PCH) \leftarrow ((SP) + 1);$   
 $(SP) \leftarrow (SP) + 2;$

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

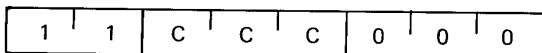


Cycles: 3  
 States: 10  
 Addressing: reg. indirect  
 Flags: none

**Rcondition** (Conditional return)

If (CCC),  
 $(PCL) \leftarrow ((SP))$   
 $(PCH) \leftarrow ((SP) + 1)$   
 $(SP) \leftarrow (SP) + 2$

If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.

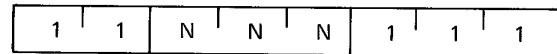


Cycles: 1/3  
 States: 5/11  
 Addressing: reg. indirect  
 Flags: none

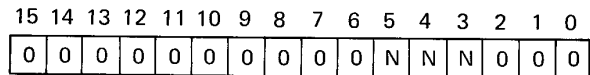
**RST n** (Restart)

$((SP) - 1) \leftarrow (PCH)$   
 $((SP) - 2) \leftarrow (PCL)$   
 $(SP) \leftarrow (SP) - 2$   
 $(PC) \leftarrow 8 * (NNN)$

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.



Cycles: 3  
 States: 11  
 Addressing: reg. indirect  
 Flags: none

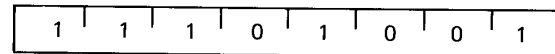


Program Counter After Restart

**PCHL** (Jump H and L indirect – move H and L to PC)

$(PCH) \leftarrow (H)$   
 $(PCL) \leftarrow (L)$

The content of register H is moved to the high-order eight bits of register PC. The content of register L is moved to the low-order eight bits of register PC.



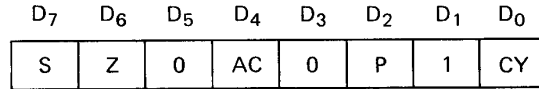
Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: none

**Stack, I/O, and Machine Control Group:**

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags.

Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

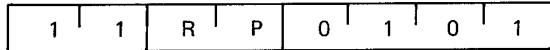
**FLAG WORD**



**PUSH rp** (Push)

$((SP) - 1) \leftarrow (rh)$   
 $((SP) - 2) \leftarrow (rl)$   
 $(SP) \leftarrow (SP) - 2$

The content of the high-order register of register pair *rp* is moved to the memory location whose address is one less than the content of register *SP*. The content of the low-order register of register pair *rp* is moved to the memory location whose address is two less than the content of register *SP*. The content of register *SP* is decremented by 2. **Note: Register pair *rp* = *SP* may not be specified.**

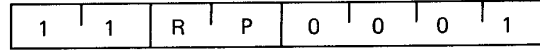


Cycles: 3  
 States: 11  
 Addressing: reg. indirect  
 Flags: none

**POP rp** (Pop)

$(rl) \leftarrow ((SP))$   
 $(rh) \leftarrow ((SP) + 1)$   
 $(SP) \leftarrow (SP) + 2$

The content of the memory location, whose address is specified by the content of register *SP*, is moved to the low-order register of register pair *rp*. The content of the memory location, whose address is one more than the content of register *SP*, is moved to the high-order register of register pair *rp*. The content of register *SP* is incremented by 2. **Note: Register pair *rp* = *SP* may not be specified.**

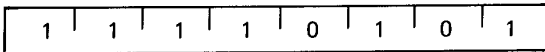


Cycles: 3  
 States: 10  
 Addressing: reg. indirect  
 Flags: none

**PUSH PSW** (Push processor status word)

$((SP) - 1) \leftarrow (A)$   
 $((SP) - 2)_0 \leftarrow (CY), ((SP) - 2)_1 \leftarrow 1$   
 $((SP) - 2)_2 \leftarrow (P), ((SP) - 2)_3 \leftarrow 0$   
 $((SP) - 2)_4 \leftarrow (AC), ((SP) - 2)_5 \leftarrow 0$   
 $((SP) - 2)_6 \leftarrow (Z), ((SP) - 2)_7 \leftarrow (S)$   
 $(SP) \leftarrow (SP) - 2$

The content of register *A* is moved to the memory location whose address is one less than register *SP*. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register *SP*. The content of register *SP* is decremented by two.

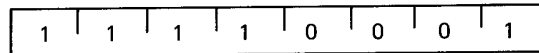


Cycles: 3  
 States: 11  
 Addressing: reg. indirect  
 Flags: none

**POP PSW** (Pop processor status word)

$(CY) \leftarrow ((SP))_0$   
 $(P) \leftarrow ((SP))_2$   
 $(AC) \leftarrow ((SP))_4$   
 $(Z) \leftarrow ((SP))_6$   
 $(S) \leftarrow ((SP))_7$   
 $(A) \leftarrow ((SP) + 1)$   
 $(SP) \leftarrow (SP) + 2$

The content of the memory location whose address is specified by the content of register *SP* is used to restore the condition flags. The content of the memory location whose address is one more than the content of register *SP* is moved to register *A*. The content of register *SP* is incremented by 2.

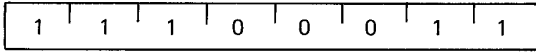


Cycles: 3  
 States: 10  
 Addressing: reg. indirect  
 Flags: Z,S,P,CY,AC

**XTHL** (Exchange stack top with H and L)

(L)  $\leftrightarrow$  ((SP))  
 (H)  $\leftrightarrow$  ((SP) + 1)

The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.

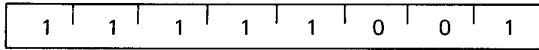


Cycles: 5  
 States: 18  
 Addressing: reg. indirect  
 Flags: none

**SPHL** (Move HL to SP)

(SP)  $\leftarrow$  (H) (L)

The contents of registers H and L (16 bits) are moved to register SP.

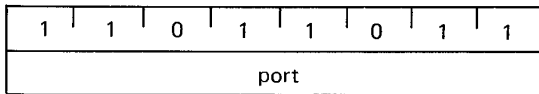


Cycles: 1  
 States: 5  
 Addressing: register  
 Flags: none

**IN port** (Input)

(A)  $\leftarrow$  (data)

The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.

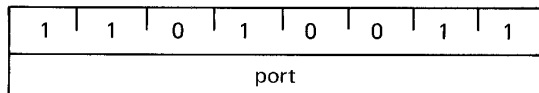


Cycles: 3  
 States: 10  
 Addressing: direct  
 Flags: none

**OUT port** (Output)

(data)  $\leftarrow$  (A)

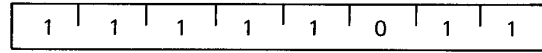
The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.



Cycles: 3  
 States: 10  
 Addressing: direct  
 Flags: none

**EI** (Enable interrupts)

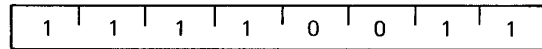
The interrupt system is enabled **following the execution of the next instruction.**



Cycles: 1  
 States: 4  
 Flags: none

**DI** (Disable interrupts)

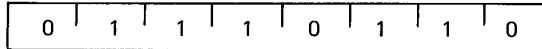
The interrupt system is disabled **immediately following the execution of the DI instruction.**



Cycles: 1  
 States: 4  
 Flags: none

**HLT** (Halt)

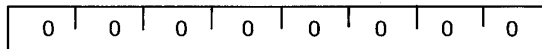
The processor is stopped. The registers and flags are unaffected.



Cycles: 1  
 States: 7  
 Flags: none

**NOP** (No op)

No operation is performed. The registers and flags are unaffected.



Cycles: 1  
 States: 4  
 Flags: none

# INSTRUCTION SET

## Summary of Processor Instructions

Mnemonic	Description	Instruction Code <sup>[1]</sup>								Clock <sup>[2]</sup> Cycles	Mnemonic	Description	Instruction Code <sup>[1]</sup>								Clock <sup>[2]</sup> Cycles
		D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>				D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>	
MOV <sub>r1,r2</sub>	Move register to register	0	1	0	0	0	S	S	S	5	RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
MOV <sub>M,r</sub>	Move register to memory	0	1	1	1	0	S	S	S	7	RNZ	Return on no zero	1	1	0	0	0	0	C	0	5/11
MOV <sub>r,M</sub>	Move memory to register	0	1	0	0	0	1	1	0	7	RP	Return on positive	1	1	1	1	0	0	0	0	5/11
HLT	Halt	0	1	1	1	0	1	1	0	7	RM	Return on minus	1	1	1	1	1	0	0	0	5/11
MVI <sub>r</sub>	Move immediate register	0	0	0	0	0	1	1	0	7	RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
MVI <sub>M</sub>	Move immediate memory	0	0	1	1	0	1	1	0	10	RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
INR <sub>r</sub>	Increment register	0	0	0	0	0	1	0	0	5	RST	Restart	1	1	A	A	A	1	1	1	11
DCR <sub>r</sub>	Decrement register	0	0	0	0	0	1	0	1	5	IN	Input	1	1	0	1	1	0	1	1	10
INR <sub>M</sub>	Increment memory	0	0	1	1	0	1	0	0	10	OUT	Output	1	1	0	1	0	0	1	1	10
DCR <sub>M</sub>	Decrement memory	0	0	1	1	0	1	0	1	10	LXI <sub>B</sub>	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
ADD <sub>r</sub>	Add register to A	1	0	0	0	0	S	S	S	4	LXI <sub>D</sub>	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
ADC <sub>r</sub>	Add register to A with carry	1	0	0	0	1	S	S	S	4	LXI <sub>H</sub>	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
SUB <sub>r</sub>	Subtract register from A	1	0	0	1	0	S	S	S	4	LXI <sub>SP</sub>	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
SBB <sub>r</sub>	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4	PUSH <sub>B</sub>	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
ANA <sub>r</sub>	And register with A	1	0	1	0	0	S	S	S	4	PUSH <sub>D</sub>	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
XRA <sub>r</sub>	Exclusive Or register with A	1	0	1	0	1	S	S	S	4	PUSH <sub>H</sub>	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
ORA <sub>r</sub>	Or register with A	1	0	1	1	0	S	S	S	4	PUSH <sub>PSW</sub>	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
CMP <sub>r</sub>	Compare register with A	1	0	1	1	1	S	S	S	4	POP <sub>B</sub>	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
ADD <sub>M</sub>	Add memory to A	1	0	0	0	0	1	1	0	7	POP <sub>D</sub>	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
ADC <sub>M</sub>	Add memory to A with carry	1	0	0	0	1	1	1	0	7	POP <sub>H</sub>	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
SUB <sub>M</sub>	Subtract memory from A	1	0	0	1	0	1	1	0	7	POP <sub>PSW</sub>	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
SBB <sub>M</sub>	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7	STA	Store A direct	0	0	1	1	0	0	1	0	13
ANA <sub>M</sub>	And memory with A	1	0	1	0	0	1	1	0	7	LDA	Load A direct	0	0	1	1	1	0	1	0	13
XRA <sub>M</sub>	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7	XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
ORA <sub>M</sub>	Or memory with A	1	0	1	1	0	1	1	0	7	XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
CMP <sub>M</sub>	Compare memory with A	1	0	1	1	1	1	1	0	7	SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7	PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7	DAD <sub>B</sub>	Add B & C to H & L	0	0	0	0	1	0	0	1	10
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7	DAD <sub>D</sub>	Add D & E to H & L	0	0	0	1	1	0	0	1	10
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7	DAD <sub>H</sub>	Add H & L to H & L	0	0	1	0	1	0	0	1	10
ANI	And immediate with A	1	1	1	0	0	1	1	0	7	DAD <sub>SP</sub>	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7	STAX <sub>B</sub>	Store A indirect	0	0	0	0	0	0	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7	STAX <sub>D</sub>	Store A indirect	0	0	0	1	0	0	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7	LDAX <sub>B</sub>	Load A indirect	0	0	0	1	0	1	0	7	
RLC	Rotate A left	0	0	0	0	0	1	1	1	4	LDAX <sub>D</sub>	Load A indirect	0	0	0	1	1	0	1	0	7
RRC	Rotate A right	0	0	0	0	1	1	1	1	4	INX <sub>B</sub>	Increment B & C registers	0	0	0	0	0	0	1	1	5
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4	INX <sub>D</sub>	Increment D & E registers	0	0	0	1	0	0	1	1	5
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4	INX <sub>H</sub>	Increment H & L registers	0	0	1	0	0	0	1	1	5
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10	INX <sub>SP</sub>	Increment stack pointer	0	0	1	1	0	0	1	1	5
JC	Jump on carry	1	1	0	1	1	0	1	0	10	DCX <sub>B</sub>	Decrement B & C	0	0	0	0	1	0	1	1	5
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10	DCX <sub>D</sub>	Decrement D & E	0	0	0	1	1	0	1	1	5
JZ	Jump on zero	1	1	0	0	1	0	1	0	10	DCX <sub>H</sub>	Decrement H & L	0	0	1	0	1	0	1	1	5
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10	DCX <sub>SP</sub>	Decrement stack pointer	0	0	1	1	1	0	1	1	5
JP	Jump on positive	1	1	1	1	0	0	1	0	10	CMA	Complement A	0	0	1	0	1	1	1	1	4
JM	Jump on minus	1	1	1	1	0	1	0	0	10	STC	Set carry	0	0	1	1	0	1	1	1	4
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10	CMC	Complement carry	0	0	1	1	1	1	1	1	4
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10	DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
CALL	Call unconditional	1	1	0	0	1	1	0	1	17	SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
CC	Call on carry	1	1	0	1	1	1	0	0	11/17	LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17	EI	Enable interrupts	1	1	1	1	0	1	1	4	
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17	DI	Disable interrupt	1	1	1	1	0	0	1	1	4
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17	NOP	No-operation	0	0	0	0	0	0	0	0	4
CP	Call on positive	1	1	1	1	0	1	0	0	11/17											
CM	Call on minus	1	1	1	1	1	1	0	0	11/17											
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17											
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17											
RET	Return	1	1	0	0	1	0	0	1	10											
RC	Return on carry	1	1	0	1	1	0	0	0	5/11											
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11											

NOTES: 1. DDD or SSS – 000 B – 001 C – 010 D – 011 E – 100 H – 101 L – 110 Memory – 111 A.  
2. Two possible cycle times, (5/11) indicate instruction cycles dependent on condition flags.



# Silicon Gate MOS 8080A

## SINGLE CHIP 8-BIT N-CHANNEL MICROPROCESSOR

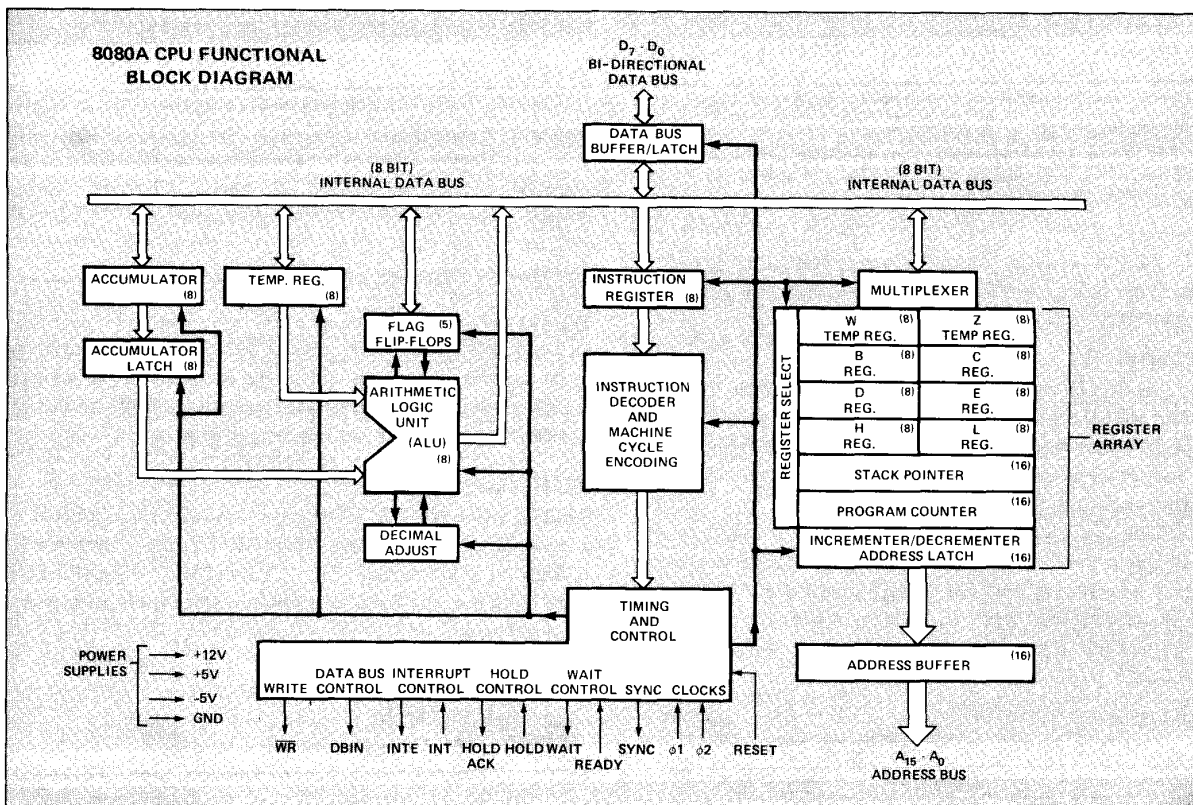
The 8080A is functionally and electrically compatible with the Intel® 8080.

- TTL Drive Capability
- 2  $\mu$ s Instruction Cycle
- Powerful Problem Solving Instruction Set
- Six General Purpose Registers and an Accumulator
- Sixteen Bit Program Counter for Directly Addressing up to 64K Bytes of Memory
- Sixteen Bit Stack Pointer and Stack Manipulation Instructions for Rapid Switching of the Program Environment
- Decimal, Binary and Double Precision Arithmetic
- Ability to Provide Priority Vectored Interrupts
- 512 Directly Addressed I/O Ports

The Intel® 8080A is a complete 8-bit parallel central processing unit (CPU). It is fabricated on a single LSI chip using Intel's n-channel silicon gate MOS process. This offers the user a high performance solution to control and processing applications. The 8080A contains six 8-bit general purpose working registers and an accumulator. The six general purpose registers may be addressed individually or in pairs providing both single and double precision operators. Arithmetic and logical instructions set or reset four testable flags. A fifth flag provides decimal arithmetic operation.

The 8080A has an external stack feature wherein any portion of memory may be used as a last in/first out stack to store/retrieve the contents of the accumulator, flags, program counter and all of the six general purpose registers. The sixteen bit stack pointer controls the addressing of this external stack. This stack gives the 8080A the ability to easily handle multiple level priority interrupts by rapidly storing and restoring processor status. It also provides almost unlimited subroutine nesting.

This microprocessor has been designed to simplify systems design. Separate 16-line address and 8-line bi-directional data buses are used to facilitate easy interface to memory and I/O. Signals to control the interface to memory and I/O are provided directly by the 8080A. Ultimate control of the address and data buses resides with the HOLD signal. It provides the ability to suspend processor operation and force the address and data buses into a high impedance state. This permits OR'ing these buses with other controlling devices for (DMA) direct memory access or multi-processor operation.



# SILICON GATE MOS 8080A

## 8080A FUNCTIONAL PIN DEFINITION

The following describes the function of all of the 8080A I/O pins. Several of the descriptions refer to internal timing periods.

### A<sub>15</sub>-A<sub>0</sub> (output three-state)

ADDRESS BUS; the address bus provides the address to memory (up to 64K 8-bit words) or denotes the I/O device number for up to 256 input and 256 output devices. A<sub>0</sub> is the least significant address bit.

### D<sub>7</sub>-D<sub>0</sub> (input/output three-state)

DATA BUS; the data bus provides bi-directional communication between the CPU, memory, and I/O devices for instructions and data transfers. Also, during the first clock cycle of each machine cycle, the 8080A outputs a status word on the data bus that describes the current machine cycle. D<sub>0</sub> is the least significant bit.

### SYNC (output)

SYNCHRONIZING SIGNAL; the SYNC pin provides a signal to indicate the beginning of each machine cycle.

### DBIN (output)

DATA BUS IN; the DBIN signal indicates to external circuits that the data bus is in the input mode. This signal should be used to enable the gating of data onto the 8080A data bus from memory or I/O.

### READY (input)

READY; the READY signal indicates to the 8080A that valid memory or input data is available on the 8080A data bus. This signal is used to synchronize the CPU with slower memory or I/O devices. If after sending an address out the 8080A does not receive a READY input, the 8080A will enter a WAIT state for as long as the READY line is low. READY can also be used to single step the CPU.

### WAIT (output)

WAIT; the WAIT signal acknowledges that the CPU is in a WAIT state.

### $\overline{WR}$ (output)

WRITE; the  $\overline{WR}$  signal is used for memory WRITE or I/O output control. The data on the data bus is stable while the  $\overline{WR}$  signal is active low ( $\overline{WR} = 0$ ).

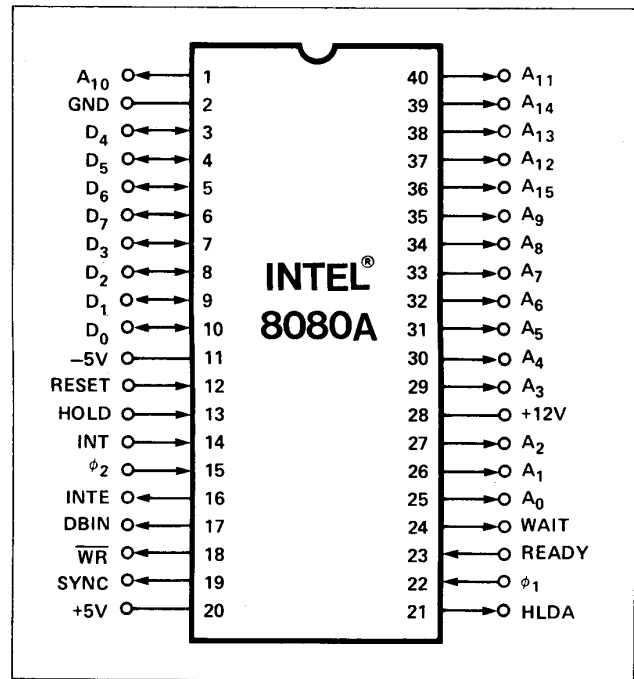
### HOLD (input)

HOLD; the HOLD signal requests the CPU to enter the HOLD state. The HOLD state allows an external device to gain control of the 8080A address and data bus as soon as the 8080A has completed its use of these buses for the current machine cycle. It is recognized under the following conditions:

- the CPU is in the HALT state.
  - the CPU is in the T<sub>2</sub> or T<sub>W</sub> state and the READY signal is active.
- As a result of entering the HOLD state the CPU ADDRESS BUS (A<sub>15</sub>-A<sub>0</sub>) and DATA BUS (D<sub>7</sub>-D<sub>0</sub>) will be in their high impedance state. The CPU acknowledges its state with the HOLD ACKNOWLEDGE (HLDA) pin.

### HLDA (output)

HOLD ACKNOWLEDGE; the HLDA signal appears in response to the HOLD signal and indicates that the data and address bus



Pin Configuration

will go to the high impedance state. The HLDA signal begins at:

- T<sub>3</sub> for READ memory or input.
- The Clock Period following T<sub>3</sub> for WRITE memory or OUTPUT operation.

In either case, the HLDA signal appears after the rising edge of  $\phi_1$  and high impedance occurs after the rising edge of  $\phi_2$ .

### INTE (output)

INTERRUPT ENABLE; indicates the content of the internal interrupt enable flip/flop. This flip/flop may be set or reset by the Enable and Disable Interrupt instructions and inhibits interrupts from being accepted by the CPU when it is reset. It is automatically reset (disabling further interrupts) at time T<sub>1</sub> of the instruction fetch cycle (M1) when an interrupt is accepted and is also reset by the RESET signal.

### INT (input)

INTERRUPT REQUEST; the CPU recognizes an interrupt request on this line at the end of the current instruction or while halted. If the CPU is in the HOLD state or if the Interrupt Enable flip/flop is reset it will not honor the request.

### RESET (input) [1]

RESET; while the RESET signal is activated, the content of the program counter is cleared. After RESET, the program will start at location 0 in memory. The INTE and HLDA flip/flops are also reset. Note that the flags, accumulator, stack pointer, and registers are not cleared.

- V<sub>SS</sub> Ground Reference.
- V<sub>DD</sub> +12 ± 5% Volts.
- V<sub>CC</sub> +5 ± 5% Volts.
- V<sub>BB</sub> -5 ± 5% Volts (substrate bias).
- $\phi_1, \phi_2$  2 externally supplied clock phases. (non TTL compatible)

# SILICON GATE MOS 8080 A

## ABSOLUTE MAXIMUM RATINGS\*

Temperature Under Bias . . . . .	0°C to +70°C
Storage Temperature . . . . .	-65°C to +150°C
All Input or Output Voltages	
With Respect to $V_{BB}$ . . . . .	-0.3V to +20V
$V_{CC}$ , $V_{DD}$ and $V_{SS}$ With Respect to $V_{BB}$	-0.3V to +20V
Power Dissipation . . . . .	1.5W

*\*COMMENT: Stresses above 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 above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## D.C. CHARACTERISTICS

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{DD} = +12\text{V} \pm 5\%$ ,  $V_{CC} = +5\text{V} \pm 5\%$ ,  $V_{BB} = -5\text{V} \pm 5\%$ ,  $V_{SS} = 0\text{V}$ , Unless Otherwise Noted.

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
$V_{ILC}$	Clock Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	$I_{OL} = 1.9\text{mA}$ on all outputs, $I_{OH} = -150\mu\text{A}$ .  Operation $T_{CY} = .48\mu\text{sec}$
$V_{IHC}$	Clock Input High Voltage	9.0		$V_{DD}+1$	V	
$V_{IL}$	Input Low Voltage	$V_{SS}-1$		$V_{SS}+0.8$	V	
$V_{IH}$	Input High Voltage	3.3		$V_{CC}+1$	V	
$V_{OL}$	Output Low Voltage			0.45	V	
$V_{OH}$	Output High Voltage	3.7			V	
$I_{DD(AV)}$	Avg. Power Supply Current ( $V_{DD}$ )		40	70	mA	
$I_{CC(AV)}$	Avg. Power Supply Current ( $V_{CC}$ )		60	80	mA	
$I_{BB(AV)}$	Avg. Power Supply Current ( $V_{BB}$ )		.01	1	mA	
$I_{IL}$	Input Leakage			$\pm 10$	$\mu\text{A}$	
$I_{CL}$	Clock Leakage			$\pm 10$	$\mu\text{A}$	$V_{SS} \leq V_{CLOCK} \leq V_{DD}$
$I_{DL}^{[2]}$	Data Bus Leakage in Input Mode			-100 -2.0	$\mu\text{A}$ mA	$V_{SS} \leq V_{IN} \leq V_{SS} + 0.8\text{V}$ $V_{SS} + 0.8\text{V} \leq V_{IN} \leq V_{CC}$
$I_{FL}$	Address and Data Bus Leakage During HOLD			+10 -100	$\mu\text{A}$	$V_{ADDR/DATA} = V_{CC}$ $V_{ADDR/DATA} = V_{SS} + 0.45\text{V}$

## CAPACITANCE

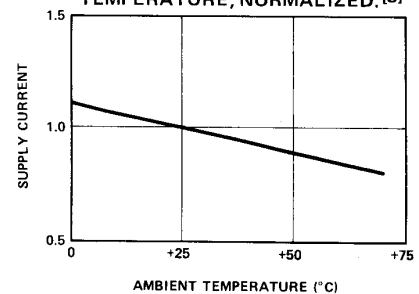
$T_A = 25^\circ\text{C}$   $V_{CC} = V_{DD} = V_{SS} = 0\text{V}$ ,  $V_{BB} = -5\text{V}$

Symbol	Parameter	Typ.	Max.	Unit	Test Condition
$C_\phi$	Clock Capacitance	17	25	pf	$f_c = 1\text{MHz}$
$C_{IN}$	Input Capacitance	6	10	pf	Unmeasured Pins
$C_{OUT}$	Output Capacitance	10	20	pf	Returned to $V_{SS}$

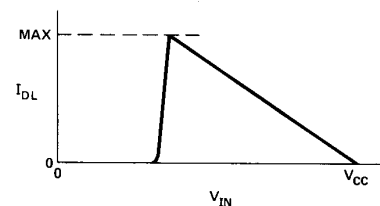
### NOTES:

1. The RESET signal must be active for a minimum of 3 clock cycles.
2. When DBIN is high and  $V_{IN} > V_{IH}$  an internal active pull up will be switched onto the Data Bus.
3.  $\Delta I_{supply} / \Delta T_A = -0.45\%/^\circ\text{C}$ .

TYPICAL SUPPLY CURRENT VS. TEMPERATURE, NORMALIZED. [3]



DATA BUS CHARACTERISTIC DURING DBIN





# SILICON GATE MOS 8080A

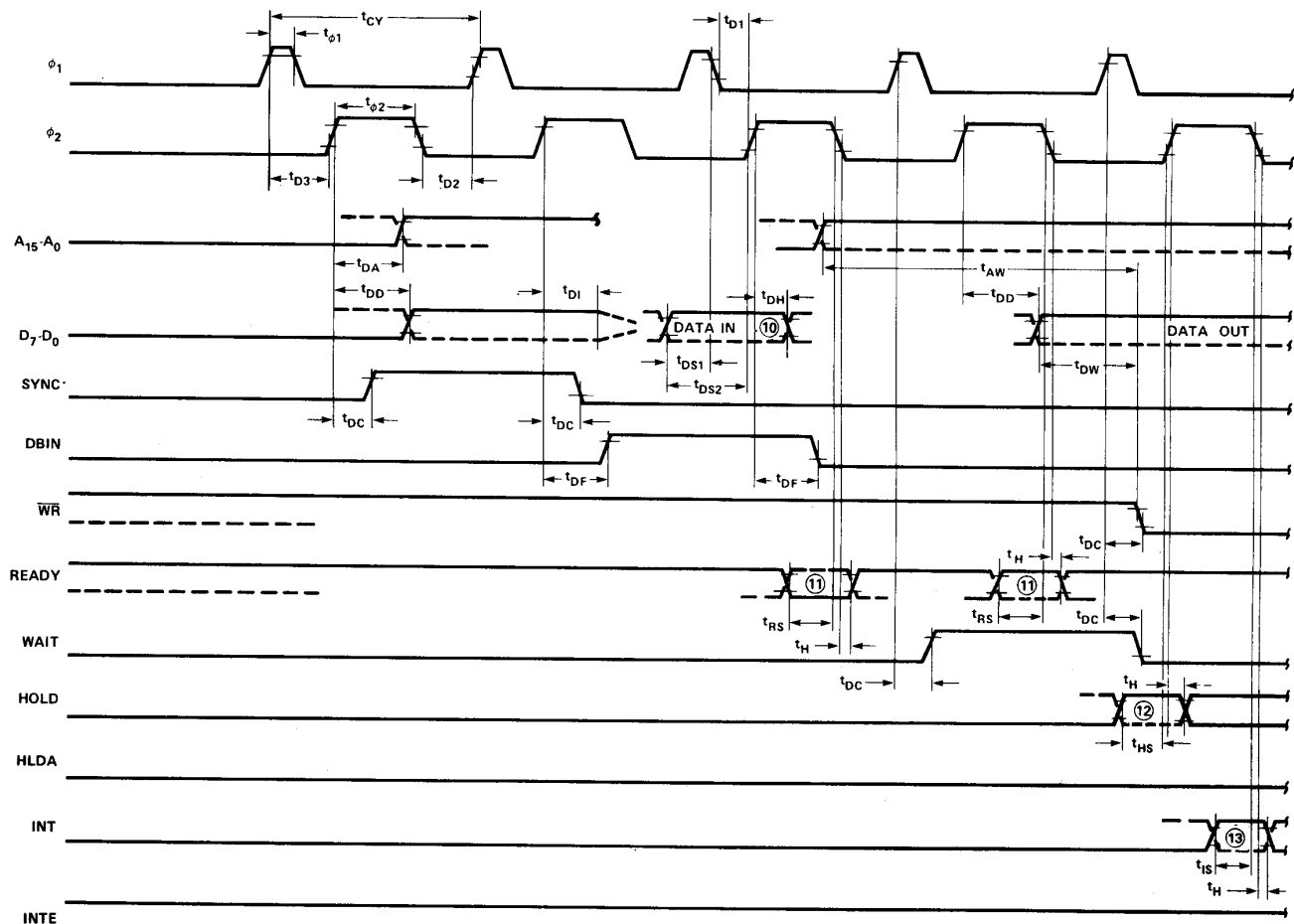
## A.C. CHARACTERISTICS

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{DD} = +12\text{V} \pm 5\%$ ,  $V_{CC} = +5\text{V} \pm 5\%$ ,  $V_{BB} = -5\text{V} \pm 5\%$ ,  $V_{SS} = 0\text{V}$ , Unless Otherwise Noted

Symbol	Parameter	Min.	Max.	Unit	Test Condition
$t_{CY}^{[3]}$	Clock Period	0.48	2.0	$\mu\text{sec}$	
$t_r, t_f$	Clock Rise and Fall Time	0	50	nsec	
$t_{\phi 1}$	$\phi_1$ Pulse Width	60		nsec	
$t_{\phi 2}$	$\phi_2$ Pulse Width	220		nsec	
$t_{D1}$	Delay $\phi_1$ to $\phi_2$	0		nsec	
$t_{D2}$	Delay $\phi_2$ to $\phi_1$	70		nsec	
$t_{D3}$	Delay $\phi_1$ to $\phi_2$ Leading Edges	80		nsec	
$t_{DA}^{[2]}$	Address Output Delay From $\phi_2$		200	nsec	$C_L = 100\text{pf}$
$t_{DD}^{[2]}$	Data Output Delay From $\phi_2$		220	nsec	
$t_{DC}^{[2]}$	Signal Output Delay From $\phi_1$ , or $\phi_2$ (SYNC, $\overline{WR}$ , WAIT, HLDA)		120	nsec	$C_L = 50\text{pf}$
$t_{DF}^{[2]}$	DBIN Delay From $\phi_2$	25	140	nsec	
$t_{DI}^{[1]}$	Delay for Input Bus to Enter Input Mode		$t_{DF}$	nsec	
$t_{DS1}$	Data Setup Time During $\phi_1$ and DBIN	30		nsec	

## TIMING WAVEFORMS <sup>[14]</sup>

(Note: Timing measurements are made at the following reference voltages: CLOCK "1" = 8.0V "0" = 1.0V; INPUTS "1" = 3.3V, "0" = 0.8V; OUTPUTS "1" = 2.0V, "0" = 0.8V.)



# SILICON GATE MOS 8080A

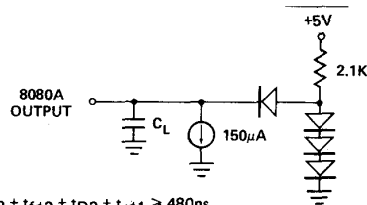
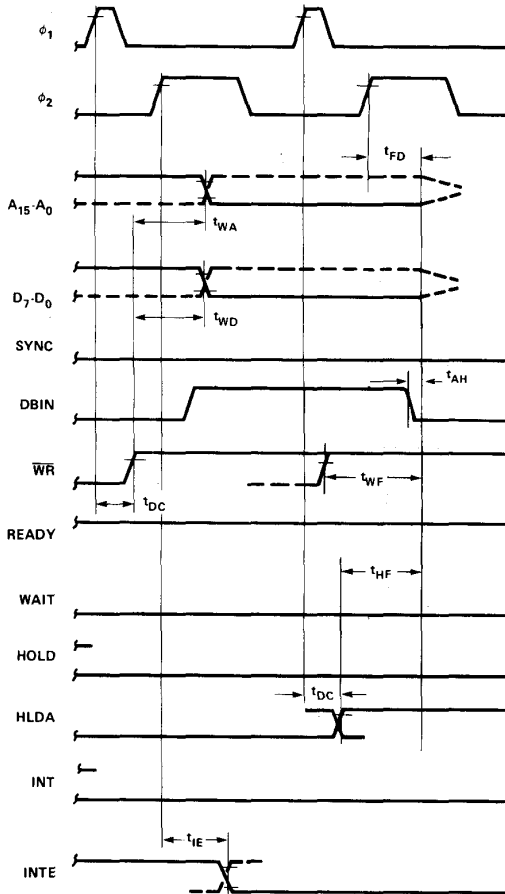
## A.C. CHARACTERISTICS (Continued)

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ,  $V_{DD} = +12\text{V} \pm 5\%$ ,  $V_{CC} = +5\text{V} \pm 5\%$ ,  $V_{BB} = -5\text{V} \pm 5\%$ ,  $V_{SS} = 0\text{V}$ , Unless Otherwise Noted

Symbol	Parameter	Min.	Max.	Unit	Test Condition	
$t_{DS2}$	Data Setup Time to $\phi_2$ During DBIN	150		nsec	$C_L = 50\text{pf}$	
$t_{DH}^{(1)}$	Data Hold Time From $\phi_2$ During DBIN	[1]		nsec		
$t_{IE}^{(2)}$	INTE Output Delay From $\phi_2$		200	nsec		
$t_{RS}$	READY Setup Time During $\phi_2$	120		nsec		
$t_{HS}$	HOLD Setup Time to $\phi_2$	140		nsec		
$t_{IS}$	INT Setup Time During $\phi_2$ (During $\phi_1$ in Halt Mode)	120		nsec		
$t_H$	Hold Time From $\phi_2$ (READY, INT, HOLD)	0		nsec		
$t_{FD}$	Delay to Float During Hold (Address and Data Bus)		120	nsec		
$t_{AW}^{(2)}$	Address Stable Prior to $\overline{WR}$	[5]		nsec		$C_L = 100\text{pf}$ : Address, Data $C_L = 50\text{pf}$ : $\overline{WR}$ , HLDA, DBIN
$t_{DW}^{(2)}$	Output Data Stable Prior to $\overline{WR}$	[6]		nsec		
$t_{WD}^{(2)}$	Output Data Stable From $\overline{WR}$	[7]		nsec		
$t_{WA}^{(2)}$	Address Stable From $\overline{WR}$	[7]		nsec		
$t_{HF}^{(2)}$	HLDA to Float Delay	[8]		nsec		
$t_{WF}^{(2)}$	$\overline{WR}$ to Float Delay	[9]		nsec		
$t_{AH}^{(2)}$	Address Hold Time After DBIN During HLDA	-20		nsec		

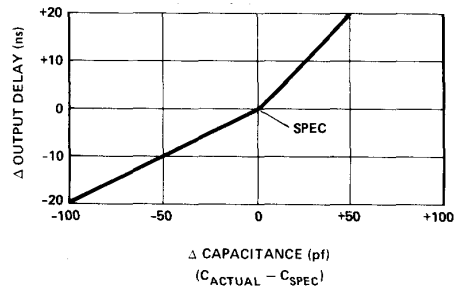
### NOTES:

- Data input should be enabled with DBIN status. No bus conflict can then occur and data hold time is assured.  $t_{DH} = 50\text{ns}$  or  $t_{DF}$ , whichever is less.
- Load Circuit.



$$3. t_{CY} = t_{D3} + t_{r\phi 2} + t_{\phi 2} + t_{f\phi 2} + t_{D2} + t_{r\phi 1} \geq 480\text{ns.}$$

### TYPICAL $\Delta$ OUTPUT DELAY VS. $\Delta$ CAPACITANCE



- The following are relevant when interfacing the 8080A to devices having  $V_{IH} = 3.3\text{V}$ :
  - Maximum output rise time from .8V to 3.3V =  $100\text{ns}$  @  $C_L = \text{SPEC}$ .
  - Output delay when measured to 3.0V =  $\text{SPEC} + 60\text{ns}$  @  $C_L = \text{SPEC}$ .
  - If  $C_L \neq \text{SPEC}$ , add .6ns/pf if  $C_L > C_{\text{SPEC}}$ , subtract .3ns/pf (from modified delay) if  $C_L < C_{\text{SPEC}}$ .
- $t_{AW} = 2 t_{CY} - t_{D3} - t_{r\phi 2} - 140\text{nsec}$ .
- $t_{DW} = t_{CY} - t_{D3} - t_{r\phi 2} - 170\text{nsec}$ .
- If not HLDA,  $t_{WD} = t_{WA} = t_{D3} + t_{r\phi 2} + 10\text{ns}$ . If HLDA,  $t_{WD} = t_{WA} = t_{WF}$ .
- $t_{HF} = t_{D3} + t_{r\phi 2} - 50\text{ns}$ .
- $t_{WF} = t_{D3} + t_{r\phi 2} - 10\text{ns}$ .
- Data in must be stable for this period during DBIN  $T_3$ . Both  $t_{DS1}$  and  $t_{DS2}$  must be satisfied.
- Ready signal must be stable for this period during  $T_2$  or  $T_W$ . (Must be externally synchronized.)
- Hold signal must be stable for this period during  $T_2$  or  $T_W$  when entering hold mode, and during  $T_3$ ,  $T_4$ ,  $T_5$  and  $T_{WH}$  when in hold mode. (External synchronization is not required.)
- Interrupt signal must be stable during this period of the last clock cycle of any instruction in order to be recognized on the following instruction. (External synchronization is not required.)
- This timing diagram shows timing relationships only; it does not represent any specific machine cycle.

# SILICON GATE MOS 8080A

## INSTRUCTION SET

The accumulator group instructions include arithmetic and logical operators with direct, indirect, and immediate addressing modes.

Move, load, and store instruction groups provide the ability to move either 8 or 16 bits of data between memory, the six working registers and the accumulator using direct, indirect, and immediate addressing modes.

The ability to branch to different portions of the program is provided with jump, jump conditional, and computed jumps. Also the ability to call to and return from sub-routines is provided both conditionally and unconditionally. The RESTART (or single byte call instruction) is useful for interrupt vector operation.

Double precision operators such as stack manipulation and double add instructions extend both the arithmetic and interrupt handling capability of the 8080A. The ability to

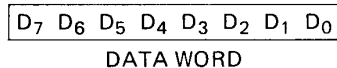
increment and decrement memory, the six general registers and the accumulator is provided as well as extended increment and decrement instructions to operate on the register pairs and stack pointer. Further capability is provided by the ability to rotate the accumulator left or right through or around the carry bit.

Input and output may be accomplished using memory addresses as I/O ports or the directly addressed I/O provided for in the 8080A instruction set.

The following special instruction group completes the 8080A instruction set: the NOP instruction, HALT to stop processor execution and the DAA instructions provide decimal arithmetic capability. STC allows the carry flag to be directly set, and the CMC instruction allows it to be complemented. CMA complements the contents of the accumulator and XCHG exchanges the contents of two 16-bit register pairs directly.

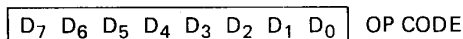
### Data and Instruction Formats

Data in the 8080A is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.



The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

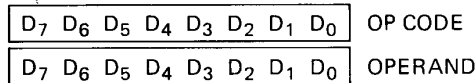
#### One Byte Instructions



#### TYPICAL INSTRUCTIONS

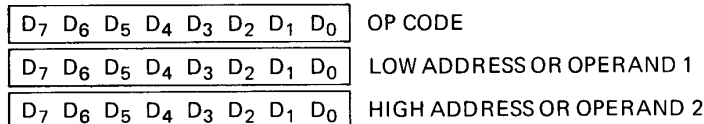
Register to register, memory reference, arithmetic or logical, rotate, return, push, pop, enable or disable  
Interrupt instructions

#### Two Byte Instructions



Immediate mode or I/O instructions

#### Three Byte Instructions



Jump, call or direct load and store instructions

For the 8080A a logic "1" is defined as a high level and a logic "0" is defined as a low level.





# Schottky Bipolar 8224

## CLOCK GENERATOR AND DRIVER FOR 8080A CPU

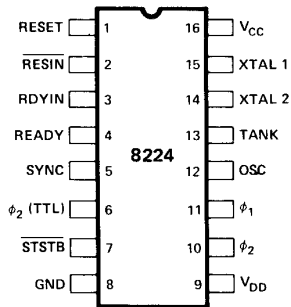
- Single Chip Clock Generator/Driver for 8080A CPU
- Power-Up Reset for CPU
- Ready Synchronizing Flip-Flop
- Advanced Status Strobe
- Oscillator Output for External System Timing
- Crystal Controlled for Stable System Operation
- Reduces System Package Count

The 8224 is a single chip clock generator/driver for the 8080A CPU. It is controlled by a crystal, selected by the designer, to meet a variety of system speed requirements.

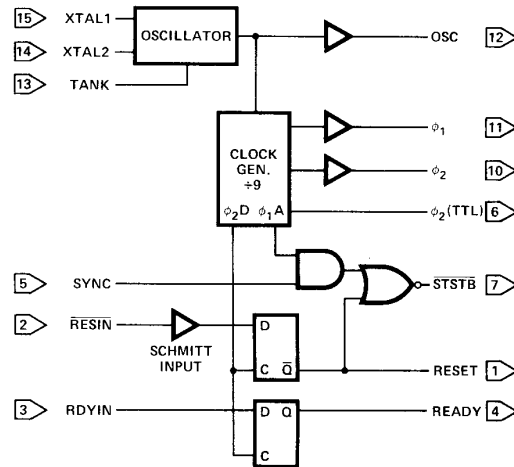
Also included are circuits to provide power-up reset, advance status strobe and synchronization of ready.

The 8224 provides the designer with a significant reduction of packages used to generate clocks and timing for 8080A.

PIN CONFIGURATION



BLOCK DIAGRAM



PIN NAMES

RESIN	RESET INPUT
RESET	RESET OUTPUT
RDYIN	READY INPUT
READY	READY OUTPUT
SYNC	SYNC INPUT
STSTB	STATUS STB (ACTIVE LOW)
phi_1	8080
phi_2	CLOCKS

XTAL 1	CONNECTIONS FOR CRYSTAL
XTAL 2	
TANK	USED WITH OVERTONE XTAL
OSC	OSCILLATOR OUTPUT
phi_2 (TTL)	phi_2 CLK (TTL LEVEL)
VCC	+5V
VDD	+12V
GND	0V

# SCHOTTKY BIPOLAR 8224

## FUNCTIONAL DESCRIPTION

### General

The 8224 is a single chip Clock Generator/Driver for the 8080A CPU. It contains a crystal-controlled oscillator, a "divide by nine" counter, two high-level drivers and several auxiliary logic functions.

### Oscillator

The oscillator circuit derives its basic operating frequency from an external, series resonant, fundamental mode crystal. Two inputs are provided for the crystal connections (XTAL1, XTAL2).

The selection of the external crystal frequency depends mainly on the speed at which the 8080A is to be run at. Basically, the oscillator operates at 9 times the desired processor speed.

A simple formula to guide the crystal selection is:

$$\text{Crystal Frequency} = \frac{1}{t_{CY}} \text{ times } 9$$

Example 1: (500ns  $t_{CY}$ )  
2mHz times 9 = 18mHz\*

Example 2: (800ns  $t_{CY}$ )  
1.25mHz times 9 = 11.25mHz

Another input to the oscillator is TANK. This input allows the use overtone mode crystals. This type of crystal generally has much lower "gain" than the fundamental type so an external LC network is necessary to provide the additional "gain" for proper oscillator operation. The external LC network is connected to the TANK input and is AC coupled to ground. See Figure 4.

The formula for the LC network is:

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

The output of the oscillator is buffered and brought out on OSC (pin 12) so that other system timing signals can be derived from this stable, crystal-controlled source.

\*When using crystals above 10mHz a small amount of frequency "trimming" may be necessary to produce the exact desired frequency. The addition of a small selected capacitance (3pF - 10pF) in series with the crystal will accomplish this function.

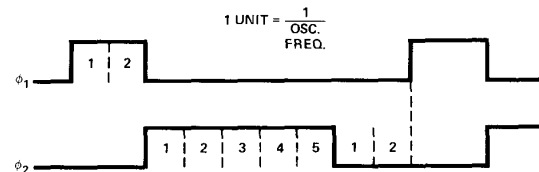
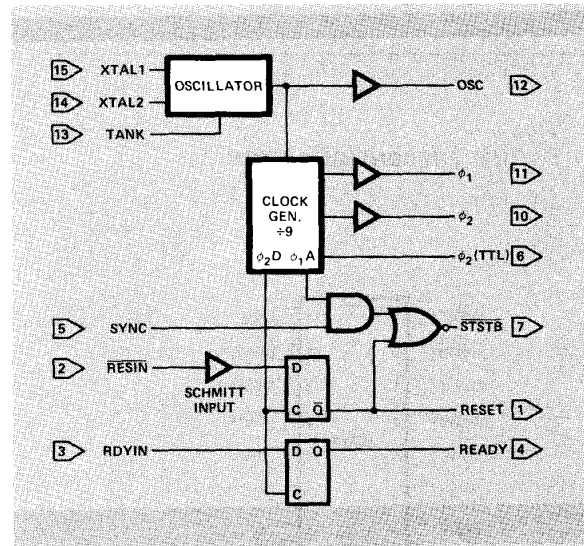
### Clock Generator

The Clock Generator consists of a synchronous "divide by nine" counter and the associated decode gating to create the waveforms of the two 8080A clocks and auxiliary timing signals.

The waveforms generated by the decode gating follow a simple 2-5-2 digital pattern. See Figure 2. The clocks generated; phase 1 and phase 2, can best be thought of as consisting of "units" based on the oscillator frequency. Assume that one "unit" equals the period of the oscillator frequency. By multiplying the number of "units" that are contained in a pulse width or delay, times the period of the oscillator frequency, the approximate time in nanoseconds can be derived.

The outputs of the clock generator are connected to two high level drivers for direct interface to the 8080A CPU. A TTL level phase 2 is also brought out  $\phi_2$  (TTL) for external timing purposes. It is especially useful in DMA dependant activities. This signal is used to gate the requesting device onto the bus once the 8080A CPU issues the Hold Acknowledgement (HLDA).

Several other signals are also generated internally so that optimum timing of the auxiliary flip-flops and status strobe ( $\overline{STSTB}$ ) is achieved.



EXAMPLE: (8080  $t_{CY}$  = 500ns)  
OSC = 18mHz/55ns  
 $\phi_1$  = 110ns (2 x 55ns)  
 $\phi_2$  = 275ns (5 x 55ns)  
 $\phi_2 - \phi_1$  = 110ns (2 x 55ns)

# SCHOTTKY BIPOLAR 8224

## STSTB (Status Strobe)

At the beginning of each machine cycle the 8080A CPU issues status information on its data bus. This information tells what type of action will take place during that machine cycle. By bringing in the SYNC signal from the CPU, and gating it with an internal timing signal ( $\phi_1A$ ), an active low strobe can be derived that occurs at the start of each machine cycle at the earliest possible moment that status data is stable on the bus. The  $\overline{STSTB}$  signal connects directly to the 8228 System Controller.

The power-on Reset also generates  $\overline{STSTB}$ , but of course, for a longer period of time. This feature allows the 8228 to be automatically reset without additional pins devoted for this function.

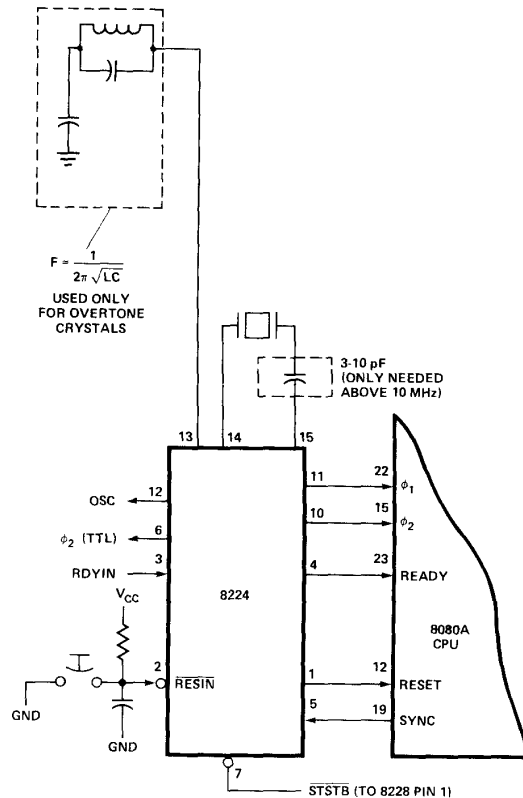
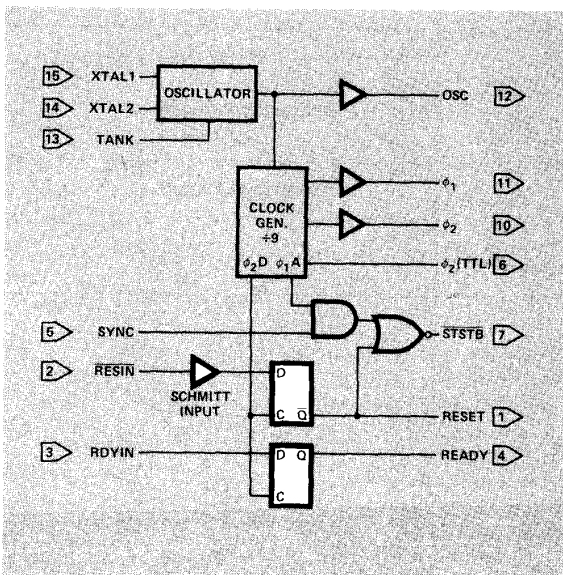
## Power-On Reset and Ready Flip-Flops

A common function in 8080A Microcomputer systems is the generation of an automatic system reset and start-up upon initial power-on. The 8224 has a built in feature to accomplish this feature.

An external RC network is connected to the  $\overline{RESIN}$  input. The slow transition of the power supply rise is sensed by an internal Schmitt Trigger. This circuit converts the slow transition into a clean, fast edge when its input level reaches a predetermined value. The output of the Schmitt Trigger is connected to a "D" type flip-flop that is clocked with  $\phi_2D$  (an internal timing signal). The flip-flop is synchronously reset and an active high level that complies with the 8080A input spec is generated. For manual switch type system Reset circuits, an active low switch closing can be connected to the  $\overline{RESIN}$  input in addition to the power-on RC network.

The READY input to the 8080A CPU has certain timing specifications such as "set-up and hold" thus, an external synchronizing flip-flop is required. The 8224 has this feature built-in. The RDYIN input presents the asynchronous "wait request" to the "D" type flip-flop. By clocking the flip-flop with  $\phi_2D$ , a synchronized READY signal at the correct input level, can be connected directly to the 8080A.

The reason for requiring an external flip-flop to synchronize the "wait request" rather than internally in the 8080 CPU is that due to the relatively long delays of MOS logic such an implementation would "rob" the designer of about 200ns during the time his logic is determining if a "wait" is necessary. An external bipolar circuit built into the clock generator eliminates most of this delay and has no effect on component count.



## SCHOTTKY BIPOLAR 8224

### D.C. Characteristics

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = +5.0\text{V} \pm 5\%$ ;  $V_{DD} = +12\text{V} \pm 5\%$ .

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$I_F$	Input Current Loading			-.25	mA	$V_F = .45\text{V}$
$I_R$	Input Leakage Current			10	$\mu\text{A}$	$V_R = 5.25\text{V}$
$V_C$	Input Forward Clamp Voltage			1.0	V	$I_C = -5\text{mA}$
$V_{IL}$	Input "Low" Voltage			.8	V	$V_{CC} = 5.0\text{V}$
$V_{IH}$	Input "High" Voltage	2.6 2.0			V	Reset Input All Other Inputs
$V_{IH}-V_{IL}$	REDIN Input Hysteresis	.25			mV	$V_{CC} = 5.0\text{V}$
$V_{OL}$	Output "Low" Voltage			.45	V	$(\phi_1, \phi_2)$ , Ready, Reset, $\overline{\text{STSTB}}$ $I_{OL} = 2.5\text{mA}$ All Other Outputs $I_{OL} = 15\text{mA}$
				.45	V	
$V_{OH}$	Output "High" Voltage $\phi_1, \phi_2$ READY, RESET All Other Outputs			9.4	V	$I_{OH} = -100\mu\text{A}$
				3.6	V	$I_{OH} = -100\mu\text{A}$
				2.4	V	$I_{OH} = -1\text{mA}$
$I_{SC}^{(1)}$	Output Short Circuit Current (All Low Voltage Outputs Only)	-10		-60	mA	$V_O = 0\text{V}$ $V_{CC} = 5.0\text{V}$
$I_{CC}$	Power Supply Current			115	mA	
$I_{DD}$	Power Supply Current			12	mA	

Note: 1. Caution,  $\phi_1$  and  $\phi_2$  output drivers do not have short circuit protection

### CRYSTAL REQUIREMENTS

Tolerance: .005% at  $0^\circ\text{C}$  -  $70^\circ\text{C}$

Resonance: Series (Fundamental)\*

Load Capacitance: 20-35pF

Equivalent Resistance: 75-20 ohms

Power Dissipation (Min): 4mW

\*With tank circuit use 3rd overtone mode.

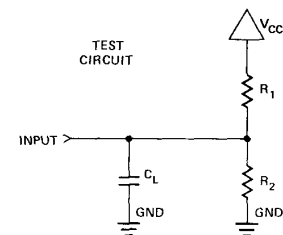


# SCHOTTKY BIPOLAR 8224

## A.C. Characteristics

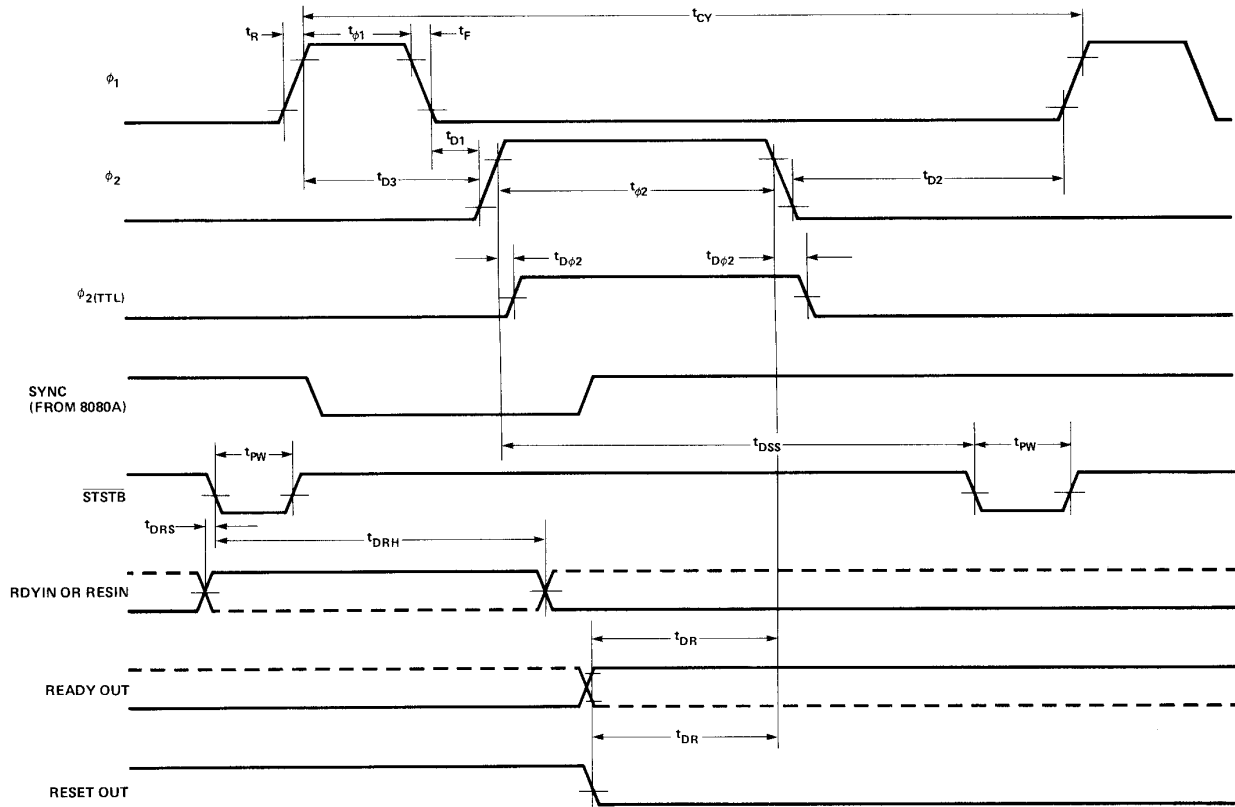
$V_{CC} = +5.0V \pm 5\%$ ;  $V_{DD} = +12.0V \pm 5\%$ ;  $T_A = 0^\circ C$  to  $70^\circ C$

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	$\phi_1$ Pulse Width	$\frac{2t_{cy}}{9} - 20ns$			ns	$C_L = 20pF$ to $50pF$
$t_{\phi 2}$	$\phi_2$ Pulse Width	$\frac{5t_{cy}}{9} - 35ns$				
$t_{D1}$	$\phi_1$ to $\phi_2$ Delay	0				
$t_{D2}$	$\phi_2$ to $\phi_1$ Delay	$\frac{2t_{cy}}{9} - 14ns$				
$t_{D3}$	$\phi_1$ to $\phi_2$ Delay	$\frac{2t_{cy}}{9}$		$\frac{2t_{cy}}{9} + 20ns$		
$t_R$	$\phi_1$ and $\phi_2$ Rise Time			20		
$t_F$	$\phi_1$ and $\phi_2$ Fall Time			20		
$t_{D\phi 2}$	$\phi_2$ to $\phi_2$ (TTL) Delay	-5		+15	ns	$\phi_2$ TTL, $C_L=30$ $R_1=300\Omega$ $R_2=600\Omega$
$t_{DSS}$	$\phi_2$ to $\overline{STSTB}$ Delay	$\frac{6t_{cy}}{9} - 30ns$		$\frac{6t_{cy}}{9}$		$\overline{STSTB}$ , $C_L=15pF$ $R_1 = 2K$ $R_2 = 4K$
$t_{PW}$	$\overline{STSTB}$ Pulse Width	$\frac{t_{cy}}{9} - 15ns$				
$t_{DRS}$	RDYIN Setup Time to Status Strobe	$50ns - \frac{4t_{cy}}{9}$				
$t_{DRH}$	RDYIN Hold Time After $\overline{STSTB}$	$\frac{4t_{cy}}{9}$				Ready & Reset $C_L=10pF$ $R_1=2K$ $R_2=4K$
$t_{DR}$	RDYIN or RESIN to $\phi_2$ Delay	$\frac{4t_{cy}}{9} - 25ns$				
$t_{CLK}$	CLK Period		$\frac{t_{cy}}{9}$			
$f_{max}$	Maximum Oscillating Frequency	27			MHz	
$C_{in}$	Input Capacitance			8	pF	$V_{CC}=+5.0V$ $V_{DD}=+12V$ $V_{BIAS}=2.5V$ $f=1MHz$



# SCHOTTKY BIPOLAR 8224

## WAVEFORMS



VOLTAGE MEASUREMENT POINTS:  $\phi_1, \phi_2$  Logic "0" = 1.0V, Logic "1" = 8.0V. All other signals measured at 1.5V.

### EXAMPLE:

#### A.C. Characteristics (For $t_{CY} = 488.28$ ns)

$T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{DD} = +5V \pm 5\%$ ;  $V_{DD} = +12V \pm 5\%$ .

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.	Max.		
$t_{\phi 1}$	$\phi_1$ Pulse Width	89			ns	$t_{CY}=488.28$ ns  $\phi_1$ & $\phi_2$ Loaded to $C_L = 20$ to $50$ pF
$t_{\phi 2}$	$\phi_2$ Pulse Width	236			ns	
$t_{D1}$	Delay $\phi_1$ to $\phi_2$	0			ns	
$t_{D2}$	Delay $\phi_2$ to $\phi_1$	95			ns	
$t_{D3}$	Delay $\phi_1$ to $\phi_2$ Leading Edges	109		129	ns	
$t_r$	Output Rise Time			20	ns	
$t_f$	Output Fall Time			20	ns	
$t_{DSS}$	$\phi_2$ to $\overline{STSTB}$ Delay	296		326	ns	Ready & Reset Loaded to $2\text{mA}/10\text{pF}$ All measurements referenced to 1.5V unless specified otherwise.
$t_{D\phi 2}$	$\phi_2$ to $\phi_2$ (TTL) Delay	-5		+15	ns	
$t_{PW}$	Status Strobe Pulse Width	40			ns	
$t_{DRS}$	RDYIN Setup Time to $\overline{STSTB}$	-167			ns	
$t_{DRH}$	RDYIN Hold Time after $\overline{STSTB}$	217			ns	
$t_{DR}$	READY or RESET to $\phi_2$ Delay	192			ns	
$f_{MAX}$	Oscillator Frequency			18.432	MHz	



# Schottky Bipolar 8228

## SYSTEM CONTROLLER AND BUS DRIVER FOR 8080A CPU

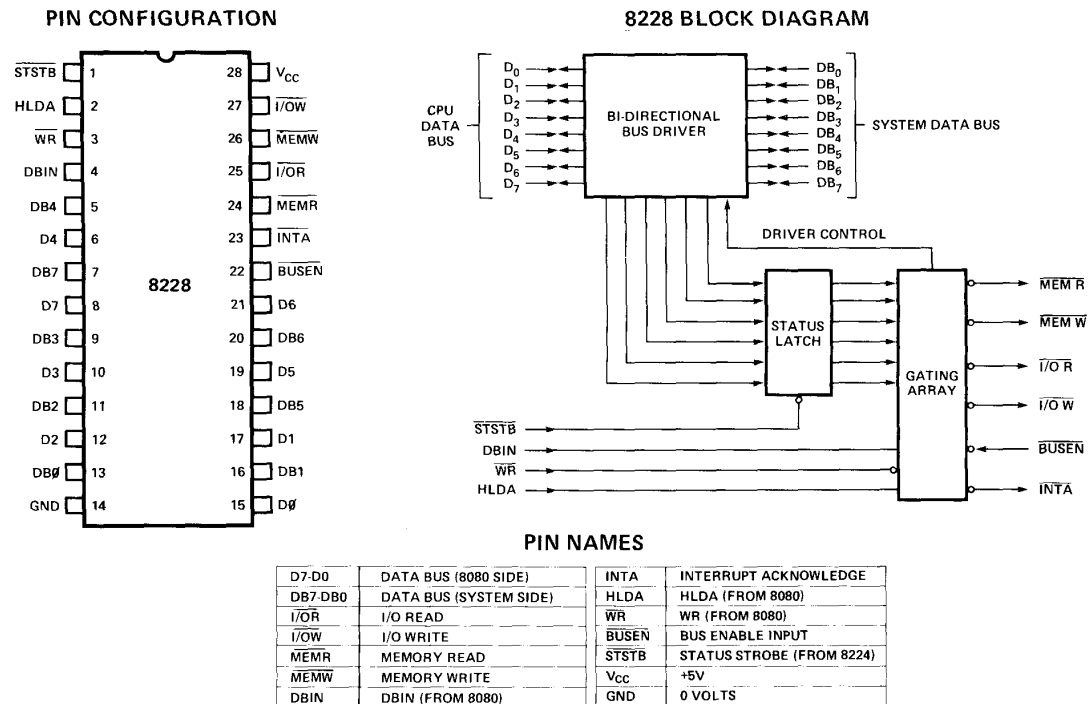
- Single Chip System Control for MCS<sup>™</sup>-80 Systems
- Built-in Bi-Directional Bus Driver for Data Bus Isolation
- Allows the use of Multiple Byte Instructions (e.g. CALL) for Interrupt Acknowledge
- User Selected Single Level Interrupt Vector (RST 7)
- 28 Pin Dual In-Line Package
- Reduces System Package Count

The 8228 is a single chip system controller and bus driver for MCS-80. It generates all signals required to directly interface MCS-80 family RAM, ROM, and I/O components.

A bi-directional bus driver is included to provide high system TTL fan-out. It also provides isolation of the 8080 data bus from memory and I/O. This allows for the optimization of control signals, enabling the systems designer to use slower memory and I/O. The isolation of the bus driver also provides for enhanced system noise immunity.

A user selected single level interrupt vector (RST 7) is provided to simplify real time, interrupt driven, small system requirements. The 8228 also generates the correct control signals to allow the use of multiple byte instructions (e.g., CALL) in response to an INTERRUPT ACKNOWLEDGE by the 8080A. This feature permits large, interrupt driven systems to have an unlimited number of interrupt levels.

The 8228 is designed to support a wide variety of system bus structures and also reduce system package count for cost effective, reliable, design of the MCS-80 systems.



# SCHOTTKY BIPOLAR 8228

## FUNCTIONAL DESCRIPTION

### General

The 8228 is a single chip System Controller and Data Bus driver for the 8080 Microcomputer System. It generates all control signals required to directly interface MCS-80™ family RAM, ROM, and I/O components.

Schottky Bipolar technology is used to maintain low delay times and provide high output drive capability to support small to medium systems.

### Bi-Directional Bus Driver

An eight bit, bi-directional bus driver is provided to buffer the 8080 data bus from Memory and I/O devices. The 8080A data bus has an input requirement of 3.3 volts (min) and can drive (sink) a maximum current of 1.9mA. The 8228 data bus driver assures that these input requirements will be not only met but exceeded for enhanced noise immunity. Also, on the system side of the driver adequate drive current is available (10mA Typ.) so that a large number of Memory and I/O devices can be directly connected to the bus.

The Bi-Directional Bus Driver is controlled by signals from the Gating Array so that proper bus flow is maintained and its outputs can be forced into their high impedance state (3-state) for DMA activities.

### Status Latch

At the beginning of each machine cycle the 8080 CPU issues "status" information on its data bus that indicates the type of activity that will occur during the cycle. The 8228 stores this information in the Status Latch when the STSTB input goes "low". The output of the Status Latch is connected to the Gating Array and is part of the Control Signal generation.

### Gating Array

The Gating Array generates control signals ( $\overline{\text{MEM R}}$ ,  $\overline{\text{MEM W}}$ ,  $\overline{\text{I/O R}}$ ,  $\overline{\text{I/O W}}$  and  $\overline{\text{INTA}}$ ) by gating the outputs of the Status Latch with signals from the 8080 CPU ( $\overline{\text{DBIN}}$ ,  $\overline{\text{WR}}$ , and  $\overline{\text{HLDA}}$ ).

The "read" control signals ( $\overline{\text{MEM R}}$ ,  $\overline{\text{I/O R}}$  and  $\overline{\text{INTA}}$ ) are derived from the logical combination of the appropriate Status Bit (or bits) and the  $\overline{\text{DBIN}}$  input from the 8080 CPU.

The "write" control signals ( $\overline{\text{MEM W}}$ ,  $\overline{\text{I/O W}}$ ) are derived from the logical combination of the appropriate Status Bit (or bits) and the  $\overline{\text{WR}}$  input from the 8080 CPU.

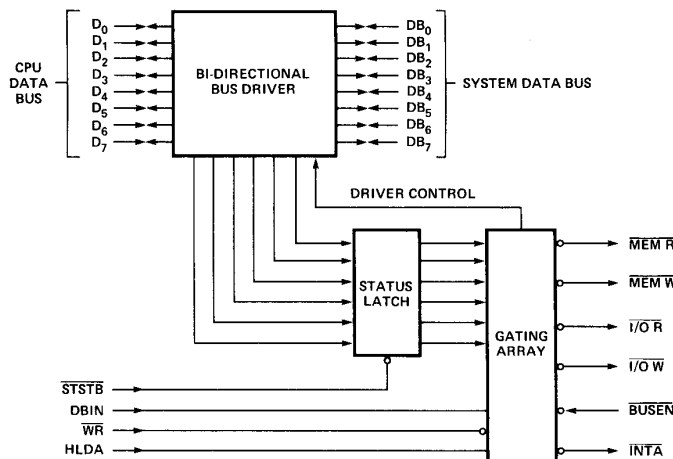
All Control Signals are "active low" and directly interface to MCS-80 family RAM, ROM and I/O components.

The  $\overline{\text{INTA}}$  control signal is normally used to gate the "interrupt instruction port" onto the bus. It also provides a special feature in the 8228. If only one basic vector is needed in the interrupt structure, such as in small systems, the 8228 can automatically insert a RST 7 instruction onto the bus at the proper time. To use this option, simply connect the  $\overline{\text{INTA}}$  output of the 8228 (pin 23) to the +12 volt supply through a series resistor (1K ohms). The voltage is sensed internally by the 8228 and logic is "set-up" so that when the  $\overline{\text{DBIN}}$  input is active a RST 7 instruction is gated on to the bus when an interrupt is acknowledged. This feature provides a single interrupt vector with no additional components, such as an interrupt instruction port.

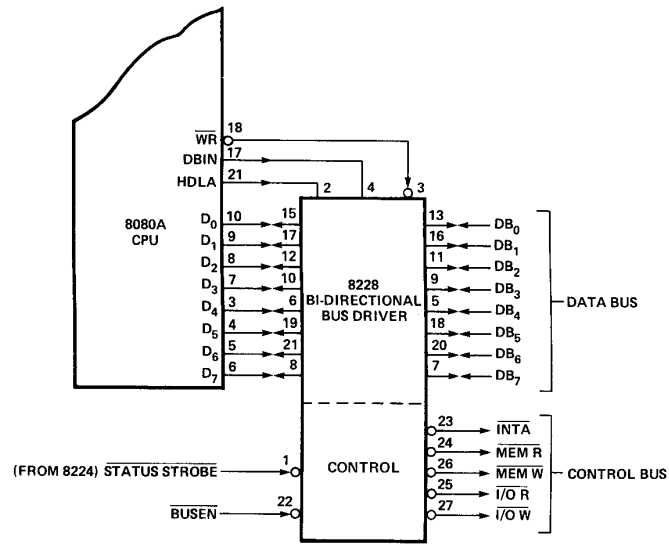
When using **CALL** as an Interrupt instruction the 8228 will generate an  $\overline{\text{INTA}}$  pulse for each of the three bytes.

The  $\overline{\text{BUSEN}}$  (Bus Enable) input to the Gating Array is an asynchronous input that forces the data bus output buffers and control signal buffers into their high-impedance state if it is a "one". If  $\overline{\text{BUSEN}}$  is a "zero" normal operation of the data buffer and control signals take place.

8228 BLOCK DIAGRAM



# SCHOTTKY BIPOLAR 8228



STATUS WORD CHART

DATA BUS BIT	STATUS INFORMATION	TYPE OF MACHINE CYCLE									
		①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
D <sub>0</sub>	INTA	0	0	0	0	0	0	0	1	0	1
D <sub>1</sub>	W <sub>0</sub>	1	1	0	1	0	1	0	1	1	1
D <sub>2</sub>	STACK	0	0	0	1	1	0	0	0	0	0
D <sub>3</sub>	HLTA	0	0	0	0	0	0	0	0	1	1
D <sub>4</sub>	OUT	0	0	0	0	0	0	1	0	0	0
D <sub>5</sub>	M <sub>1</sub>	1	0	0	0	0	0	0	1	0	1
D <sub>6</sub>	INP	0	0	0	0	0	1	0	0	0	0
D <sub>7</sub>	MEMR	1	1	0	1	0	0	0	0	1	0

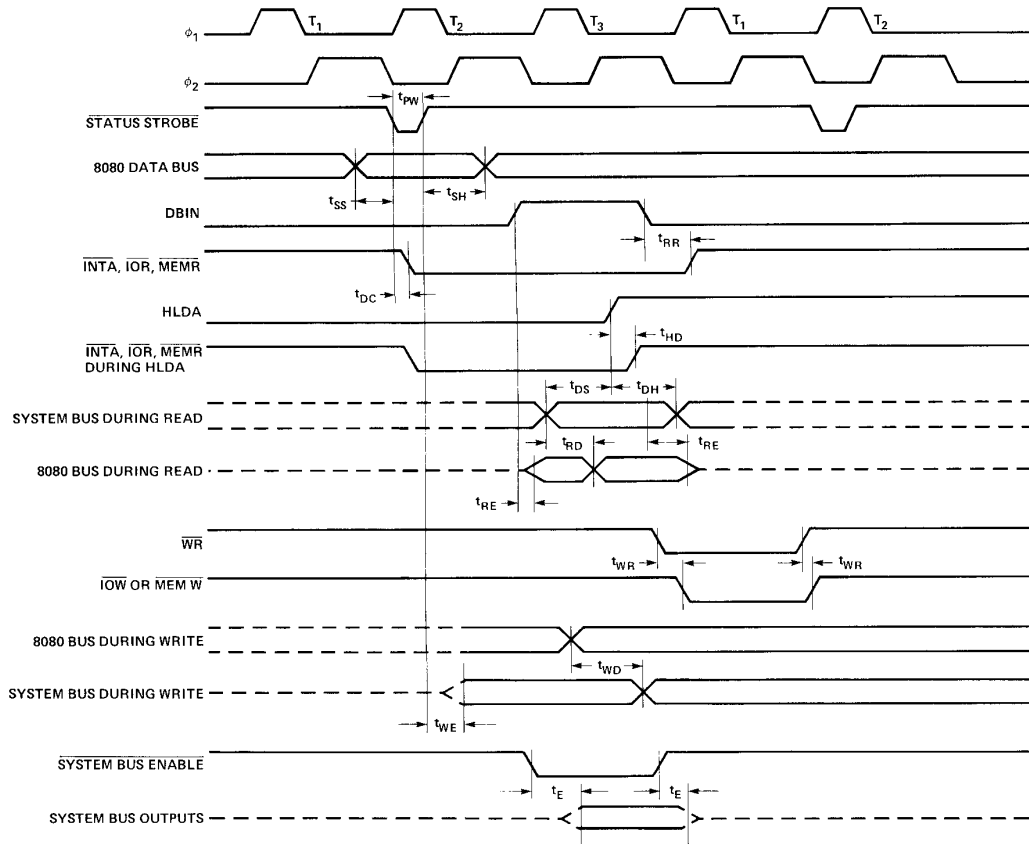
(N) STATUS WORD

CONTROL SIGNALS

- INTA
- (NONE)
- INTA
- I/O W
- I/O R
- MEM W
- MEM R
- MEM W
- MEM R
- MEM R

# SCHOTTKY BIPOLAR 8228

## WAVEFORMS



VOLTAGE MEASUREMENT POINTS: D<sub>0</sub>-D<sub>7</sub> (when outputs) Logic "0" = 0.8V, Logic "1" = 3.0V. All other signals measured at 1.5V.

### A.C. Characteristics $T_A = 0^\circ\text{C}$ to $70^\circ\text{C}$ ; $V_{CC} = 5V \pm 5\%$ .

Symbol	Parameter	Limits		Units	Condition
		Min.	Max.		
$t_{pw}$	Width of Status Strobe	22		ns	
$t_{ss}$	Setup Time, Status Inputs D <sub>0</sub> -D <sub>7</sub>	8		ns	
$t_{sh}$	Hold Time, Status Inputs D <sub>0</sub> -D <sub>7</sub>	5		ns	
$t_{DC}$	Delay from $\overline{STSTB}$ to any Control Signal	20	60	ns	$C_L = 100\text{pF}$
$t_{RR}$	Delay from DBIN to Control Outputs		30	ns	$C_L = 100\text{pF}$
$t_{RE}$	Delay from DBIN to Enable/Disable 8080 Bus		45	ns	$C_L = 25\text{pF}$
$t_{RD}$	Delay from System Bus to 8080 Bus during Read		30	ns	$C_L = 25\text{pF}$
$t_{WR}$	Delay from $\overline{WR}$ to Control Outputs	5	45	ns	$C_L = 100\text{pF}$
$t_{WE}$	Delay to Enable System Bus DB <sub>0</sub> -DB <sub>7</sub> after $\overline{STSTB}$		30	ns	$C_L = 100\text{pF}$
$t_{WD}$	Delay from 8080 Bus D <sub>0</sub> -D <sub>7</sub> to System Bus DB <sub>0</sub> -DB <sub>7</sub> during Write	5	40	ns	$C_L = 100\text{pF}$
$t_E$	Delay from System Bus Enable to System Bus DB <sub>0</sub> -DB <sub>7</sub>		30	ns	$C_L = 100\text{pF}$
$t_{HD}$	HLDA to Read Status Outputs		25	ns	
$t_{DS}$	Setup Time, System Bus Inputs to HLDA	10		ns	
$t_{DH}$	Hold Time, System Bus Inputs to HLDA	20		ns	$C_L = 100\text{pF}$

# SCHOTTKY BIPOLAR 8228

D.C. Characteristics  $T_A = 0^\circ\text{C}$  to  $70^\circ\text{C}$ ;  $V_{CC} = 5\text{V} \pm 5\%$ .

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.[1]	Max.		
$V_C$	Input Clamp Voltage, All Inputs		.75	-1.0	V	$V_{CC}=4.75\text{V}; I_C=-5\text{mA}$
$I_F$	Input Load Current, STSTB			500	$\mu\text{A}$	$V_{CC} = 5.25\text{V}$ $V_F = 0.45\text{V}$
	$D_2$ & $D_6$			750	$\mu\text{A}$	
	$D_0, D_1, D_4, D_5,$ & $D_7$			250	$\mu\text{A}$	
	All Other Inputs			250	$\mu\text{A}$	
$I_R$	Input Leakage Current STSTB			100	$\mu\text{A}$	$V_{CC} = 5.25\text{V}$ $V_R = 5.25\text{V}$
	$DB_0$ - $DB_7$			20	$\mu\text{A}$	
	All Other Inputs			100	$\mu\text{A}$	
$V_{TH}$	Input Threshold Voltage, All Inputs	0.8		2.0	V	$V_{CC} = 5\text{V}$
$I_{CC}$	Power Supply Current		140	190	mA	$V_{CC}=5.25\text{V}$
$V_{OL}$	Output Low Voltage, $D_0$ - $D_7$			.45	V	$V_{CC}=4.75\text{V}; I_{OL}=2\text{mA}$
	All Other Outputs			.45	V	$I_{OL} = 10\text{mA}$
$V_{OH}$	Output High Voltage, $D_0$ - $D_7$	3.6	3.8		V	$V_{CC}=4.75\text{V}; I_{OH}=-10\mu\text{A}$
	All Other Outputs	2.4			V	$I_{OH} = -1\text{mA}$
$I_{OS}$	Short Circuit Current, All Outputs	15		90	mA	$V_{CC}=5\text{V}$
$I_{O(off)}$	Off State Output Current, All Control Outputs			100	$\mu\text{A}$	$V_{CC}=5.25\text{V}; V_O=5.25$
				-100	$\mu\text{A}$	$V_O=.45\text{V}$
$I_{INT}$	INTA Current			5	mA	(See Figure below)

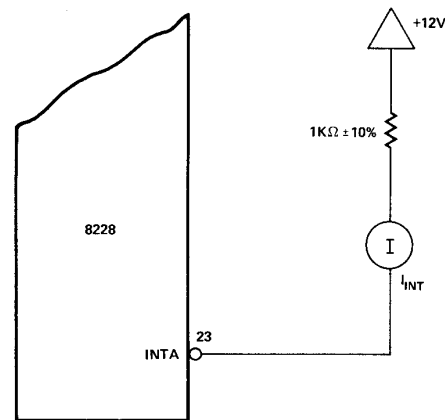
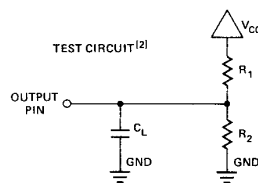
Note 1: Typical values are for  $T_A = 25^\circ\text{C}$  and nominal supply voltages.

**Capacitance** This parameter is periodically sampled and not 100% tested.

Symbol	Parameter	Limits			Unit
		Min.	Typ.[1]	Max.	
$C_{IN}$	Input Capacitance		8	12	pF
$C_{OUT}$	Output Capacitance Control Signals		7	15	pF
I/O	I/O Capacitance (D or DB)		8	15	pF

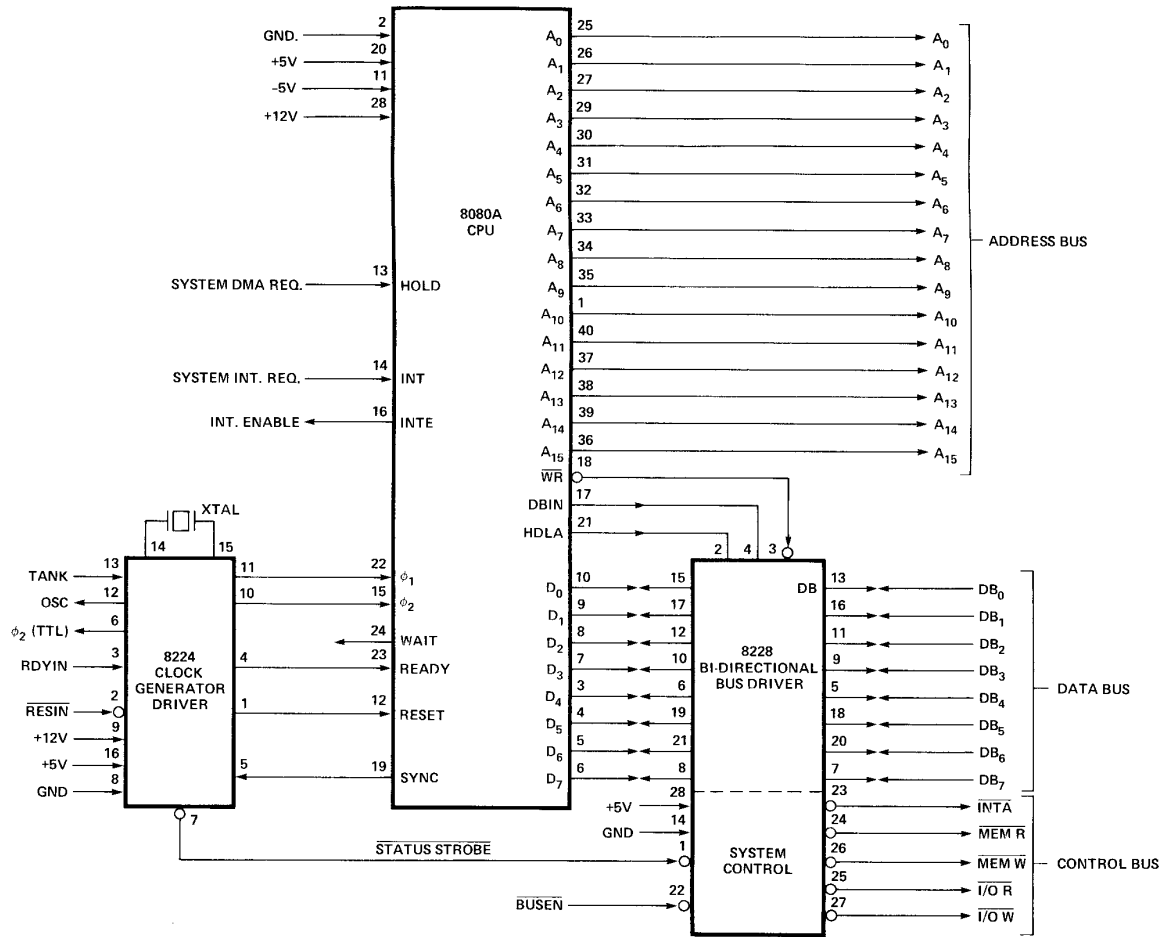
TEST CONDITIONS:  $V_{BIAS} = 2.5\text{V}$ ,  $V_{CC} = 5.0\text{V}$ ,  $T_A = 25^\circ\text{C}$ ,  $f = 1\text{MHz}$ .

Note 2: For  $D_0$ - $D_7$ :  $R_1 = 4\text{K}\Omega$ ,  $R_2 = \infty\Omega$ ,  $C_L = 25\text{pF}$ . For all other outputs:  $R_1 = 500\Omega$ ,  $R_2 = 1\text{K}\Omega$ ,  $C_L = 100\text{pF}$ .



INTA Test Circuit (for RST 7)

# SCHOTTKY BIPOLAR 8228



8080A CPU Standard Interface