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_0-D_7). Memory and peripheral device addresses are transmitted over a separate 16-bit 3-state Address Bus (A_0-A_{15}). Six timing and control outputs (SYNC, DBIN, WAIT, $\overline{WR}$, 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 (ϕ_1 and ϕ_2) are accepted by the 8080.

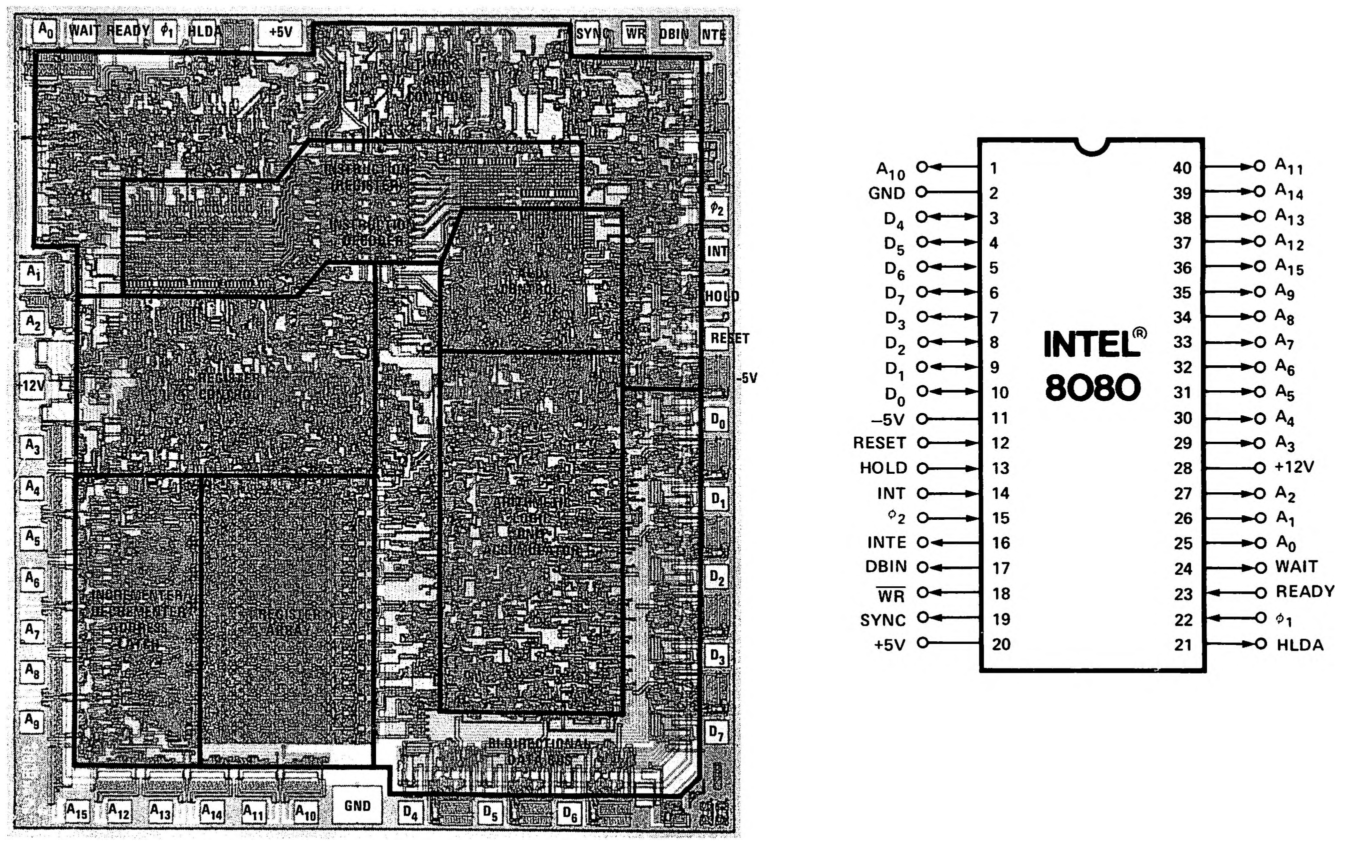


Figure 2-1. 8080 Photomicrograph With Pin Designations

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 automatically 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_0-A_{15}), 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.

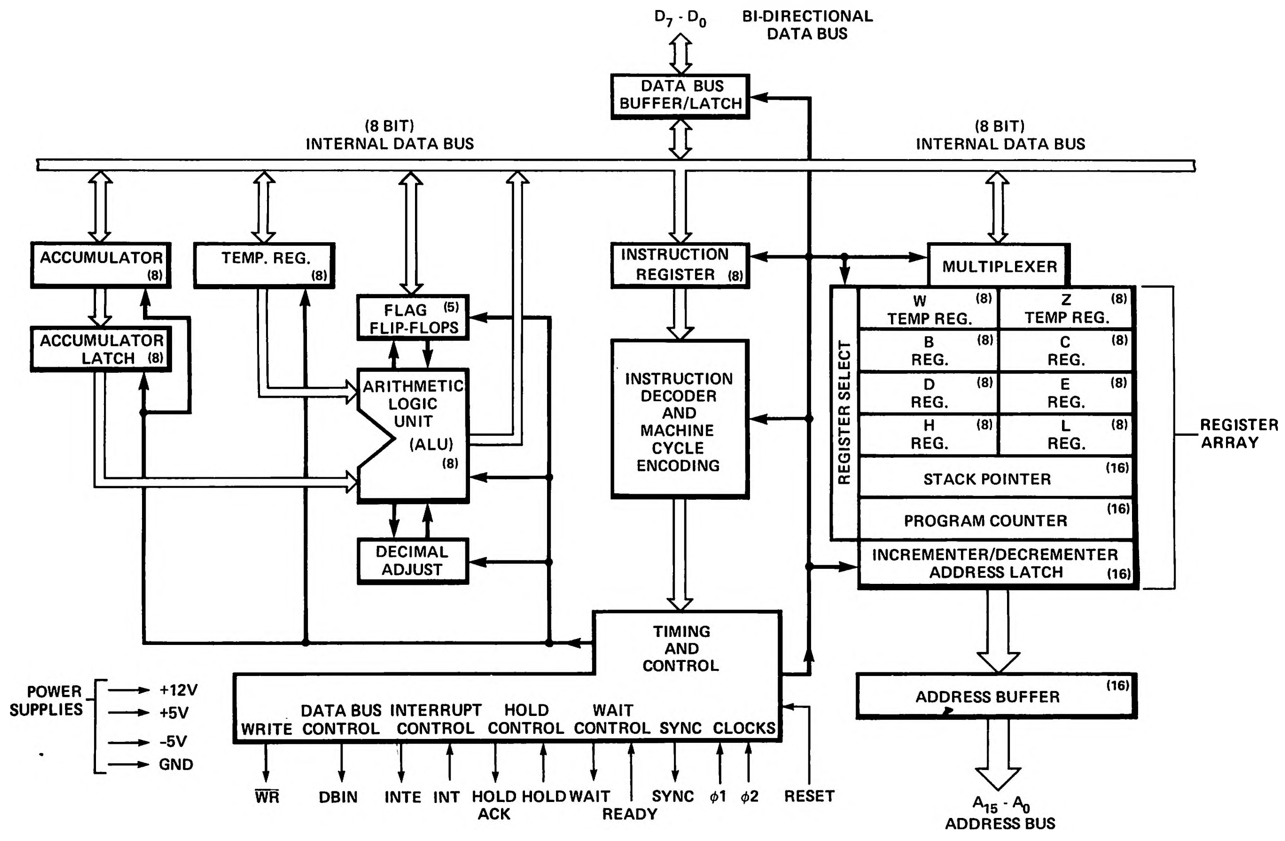


Figure 2-2. 8080 CPU Functional Block Diagram

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_0 through D_7). 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.

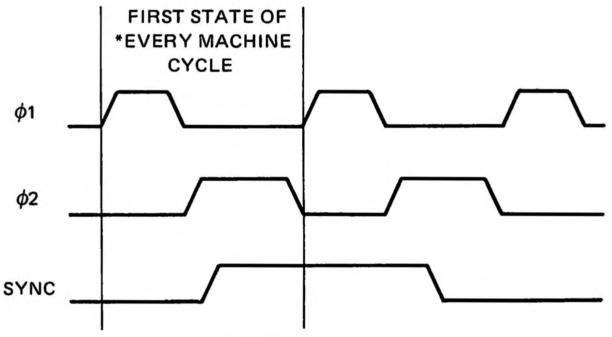
During the input mode, data from the external data bus is transferred to the internal bus. The internal bus is precharged at the beginning of each internal state, except for the transfer state (T_3—described later in this chapter).

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 ϕ_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 ϕ_1 and ϕ_2, are furnished by external circuitry. It is the ϕ_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 ϕ_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. ϕ_1, ϕ_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 eighteen 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 then 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_0-D_7), 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_1 status bit (D_6), 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_1, T_2, T_3, T_4, T_5 or T_W). 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 ϕ_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 ϕ_1 and ϕ_2 clock pulses.

The SYNC signal identifies the first state (T_1) in every machine cycle. As shown in Figure 2-5, the SYNC signal is related to the leading edge of the ϕ_2 clock. There is a delay (t_{DC}) between the low-to-high transition of ϕ_2 and the positive-going edge of the SYNC pulse. There also is a corresponding delay (also t_{DC}) between the next ϕ_2 pulse and the falling edge of the SYNC signal. Status information is displayed on D_0-D_7 during the same ϕ_2 to ϕ_2 interval. Switching of the status signals is likewise controlled by ϕ_2.

The rising edge of ϕ_2 during T_1 also loads the processor's address lines (A_0-A15). These lines become stable within a brief delay (t_{DA}) of the ϕ_2 clocking pulse, and they remain stable until the first ϕ_2 pulse after state T_3. 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 ϕ_2 pulse within state T_2 or T_W. 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_W) at the end of T_2, rather than proceeding directly to the T_3 state. Entry into the T_W 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 ϕ_1 clock and occurs within a brief delay (t_{DC}) of the actual entry into the T_W state.

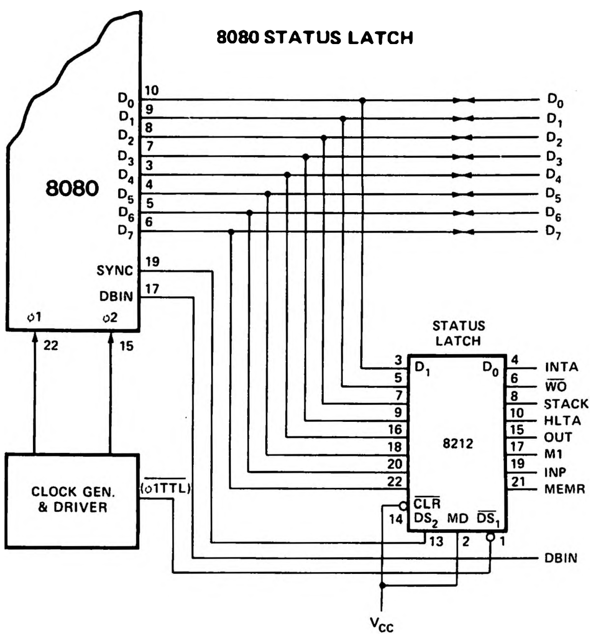
A wait period may be of indefinite duration. The processor remains in the waiting condition until its READY line again goes high. A READY indication **must** precede the falling edge of the ϕ_2 clock by a specified interval (t_{RS}), in order to guarantee an exit from the T_W state. The cycle may then proceed, beginning with the rising edge of the next ϕ_1 clock. A WAIT interval will therefore consist of an integral number of T_W states and will always be a multiple of the clock period.

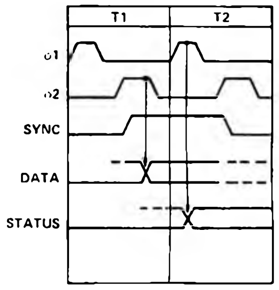
Instructions for the 8080 require from one to five machine cycles for complete execution. The 8080 sends out 8 bit of status information on the data bus at the beginning of each machine cycle (during SYNC time). The following table defines the status information.

STATUS INFORMATION DEFINITION

Symbols	Data Bus Bit	Definition
INTA*	D_0	Acknowledge signal for INTERRUPT request. Signal should be used to gate a restart instruction onto the data bus when DBIN is active.
$\overline{WO}$	D_1	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_2	Indicates that the address bus holds the pushdown stack address from the Stack Pointer.
HLTA	D_3	Acknowledge signal for HALT instruction.
OUT	D_4	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_1	D_5	Provides a signal to indicate that the CPU is in the fetch cycle for the first byte of an instruction.
INP*	D_6	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_7	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.





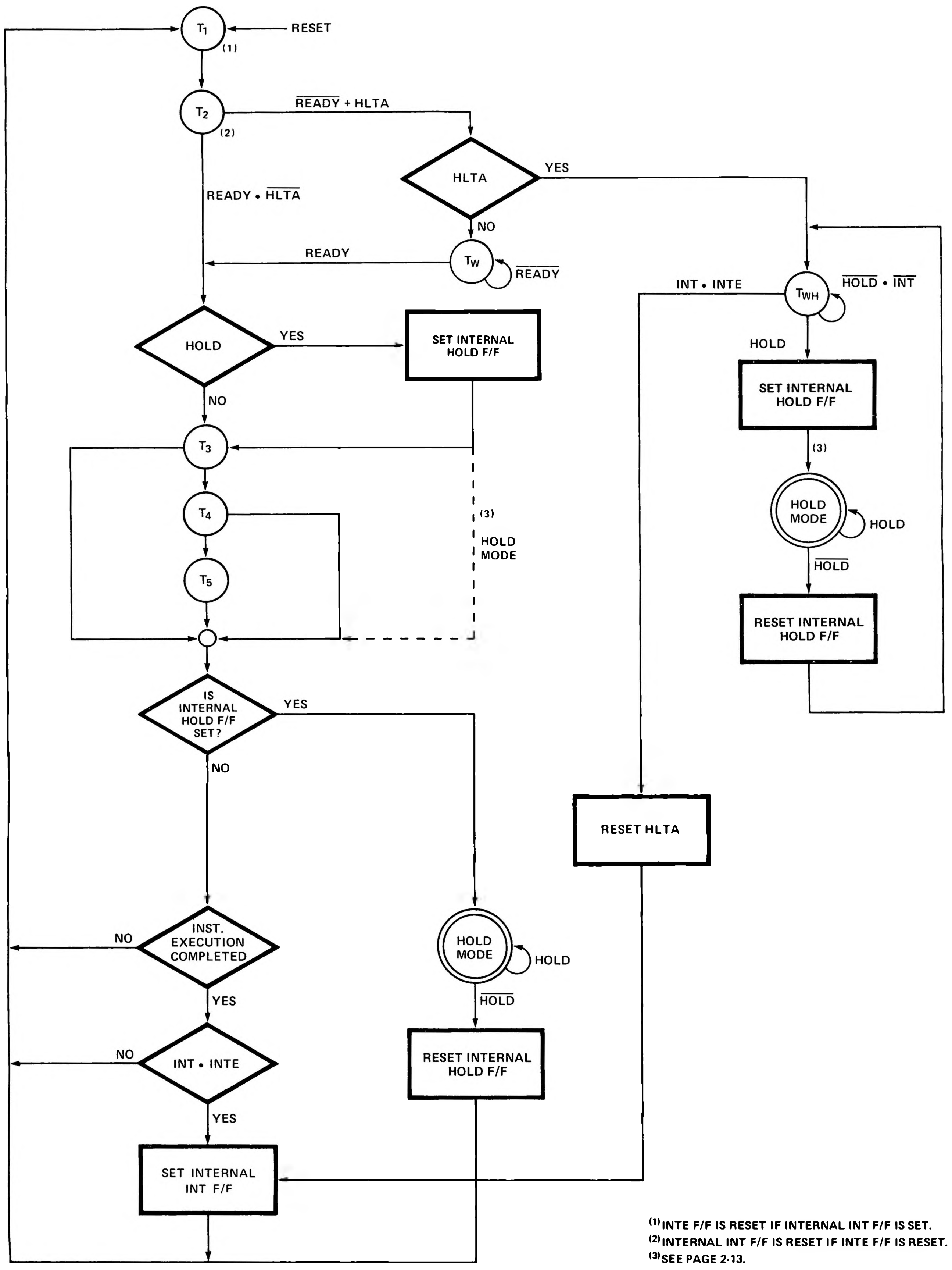
STATUS WORD CHART

TYPE OF MACHINE CYCLE

Ⓝ STATUS WORD

DATA BUS BIT	STATUS INFORMATION	① INSTRUCTION FETCH	② MEMORY READ	③ MEMORY WRITE	④ STACK READ	⑤ STACK WRITE	⑥ INPUT READ	⑦ OUTPUT WRITE	⑧ INTERRUPT ACKNOWLEDGE	⑨ HALT ACKNOWLEDGE	⑩ INTERRUPT ACKNOWLEDGE WHILE HALT
D_0	INTA	0	0	0	0	0	0	0	1	0	1
D_1	$\overline{WO}$	1	1	0	1	0	1	0	1	1	1
D_2	STACK	0	0	0	1	1	0	0	0	0	0
D_3	HLTA	0	0	0	0	0	0	0	0	1	1
D_4	OUT	0	0	0	0	0	0	1	0	0	0
D_5	M_1	1	0	0	0	0	0	0	1	0	1
D_6	INP	0	0	0	0	0	1	0	0	0	0
D_7	MEMR	1	1	0	1	0	0	0	0	1	0

Table 2-1. 8080 Status Bit Definitions



(1) INTE F/F IS RESET IF INTERNAL INT F/F IS SET.
(2) INTERNAL INT F/F IS RESET IF INTE F/F IS RESET.
(3) SEE PAGE 2-13.

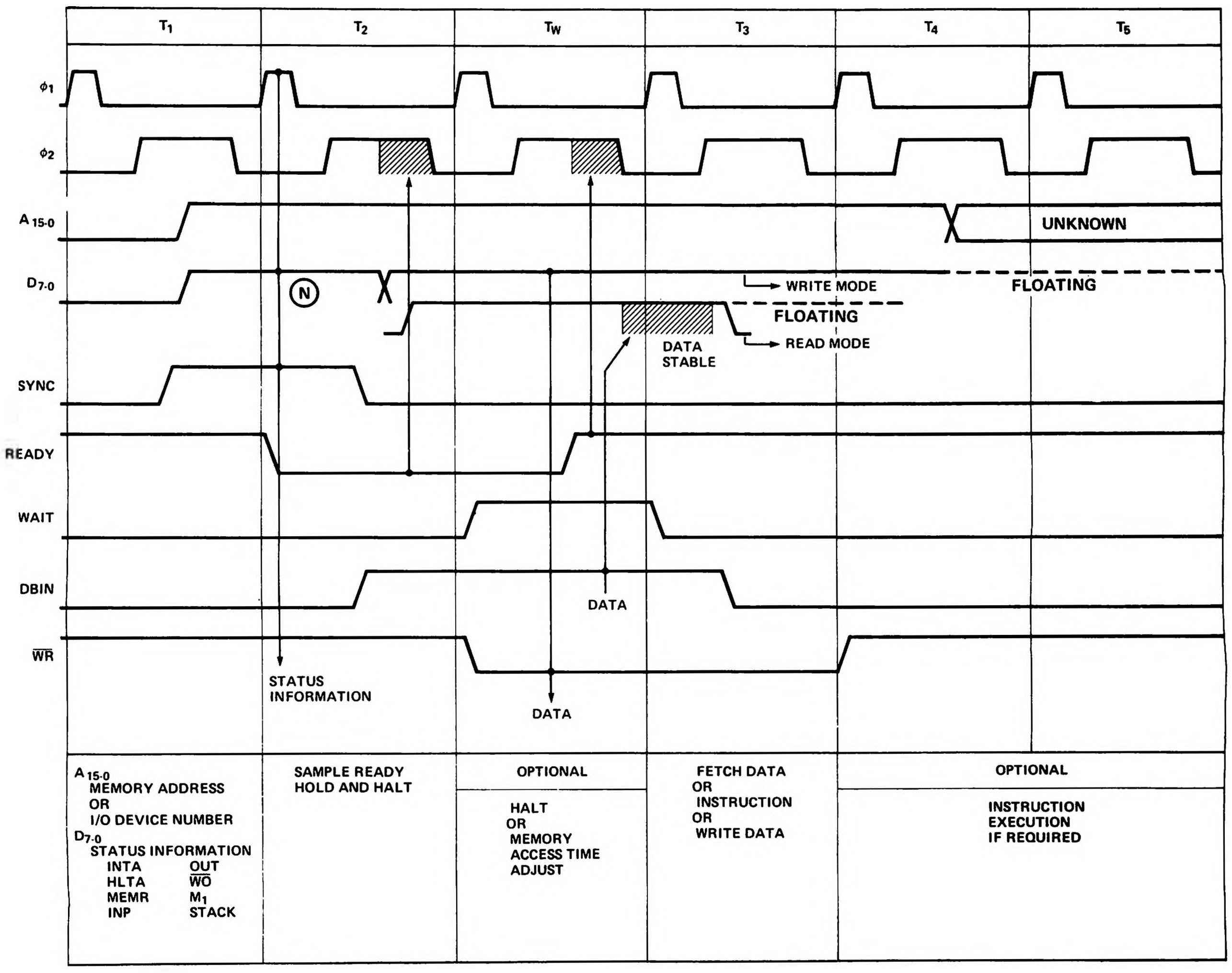
Figure 2-4. CPU State Transition Diagram

The events that take place during the T_3 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 ϕ_2 during T_2 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 "ϕ_1–data set-up" interval (t_{DS1}), that precedes the falling edge of the ϕ_1 pulse defining state T_3, and the "ϕ_2–data set-up" interval (t_{DS2}), that precedes the rising edge of ϕ_2 in state T_3. This same data must remain stable during the "data hold" interval (t_{DH}) that occurs following the rising edge of the ϕ_2 pulse. Data placed on these lines by memory or by other external devices will be sampled during T_3.

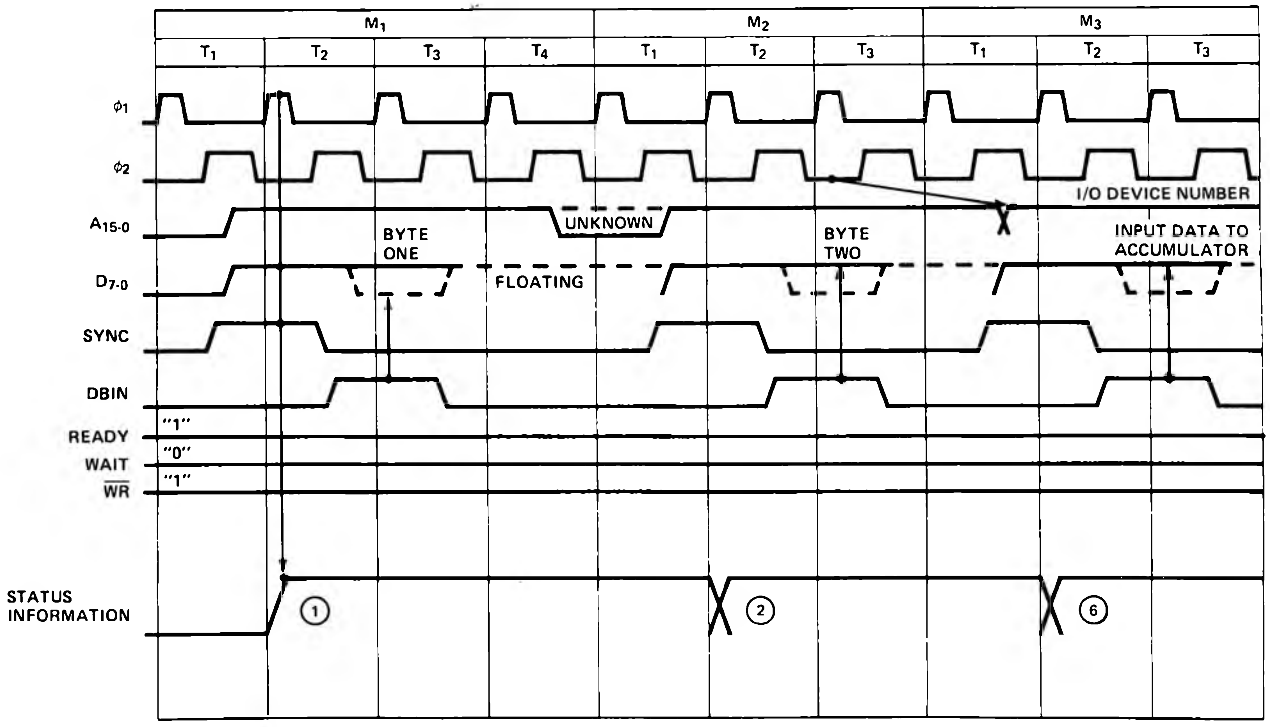
During the input of data to the processor, the 8080 generates a DBIN signal which should be used externally to enable the transfer. Machine cycles in which DBIN is available include: FETCH, MEMORY READ, STACK READ, and INTERRUPT. DBIN is initiated by the rising edge of ϕ_2 during state T2 and terminated by the corresponding edge of ϕ_2 during T_3. Any T_W phases intervening between T_2 and T_3 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 ϕ_2 within state T_2 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



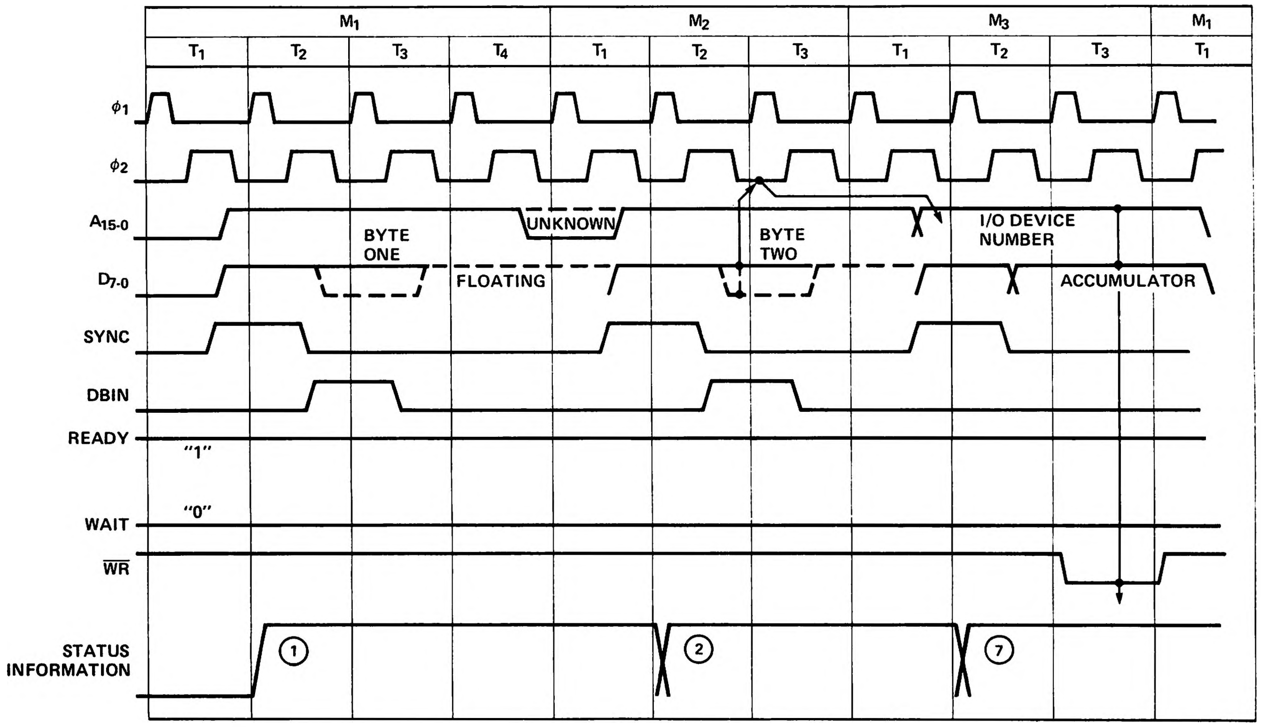
NOTE: (N) Refer to Status Word Chart on Page 2-6.

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 ϕ_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 ϕ_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 ϕ_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 necessarily extend $\overline{WR}$, in much the same way that 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 ($A_{15\text{-}0}$); status information is placed on Data Bus ($D_{7\text{-}0}$).
T_2	The CPU samples the READY and HOLD inputs and checks for halt instruction.
TW (optional)	Processor enters wait state if READY is low or if HALT instruction has been executed.
T3	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.
T4 T5 (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

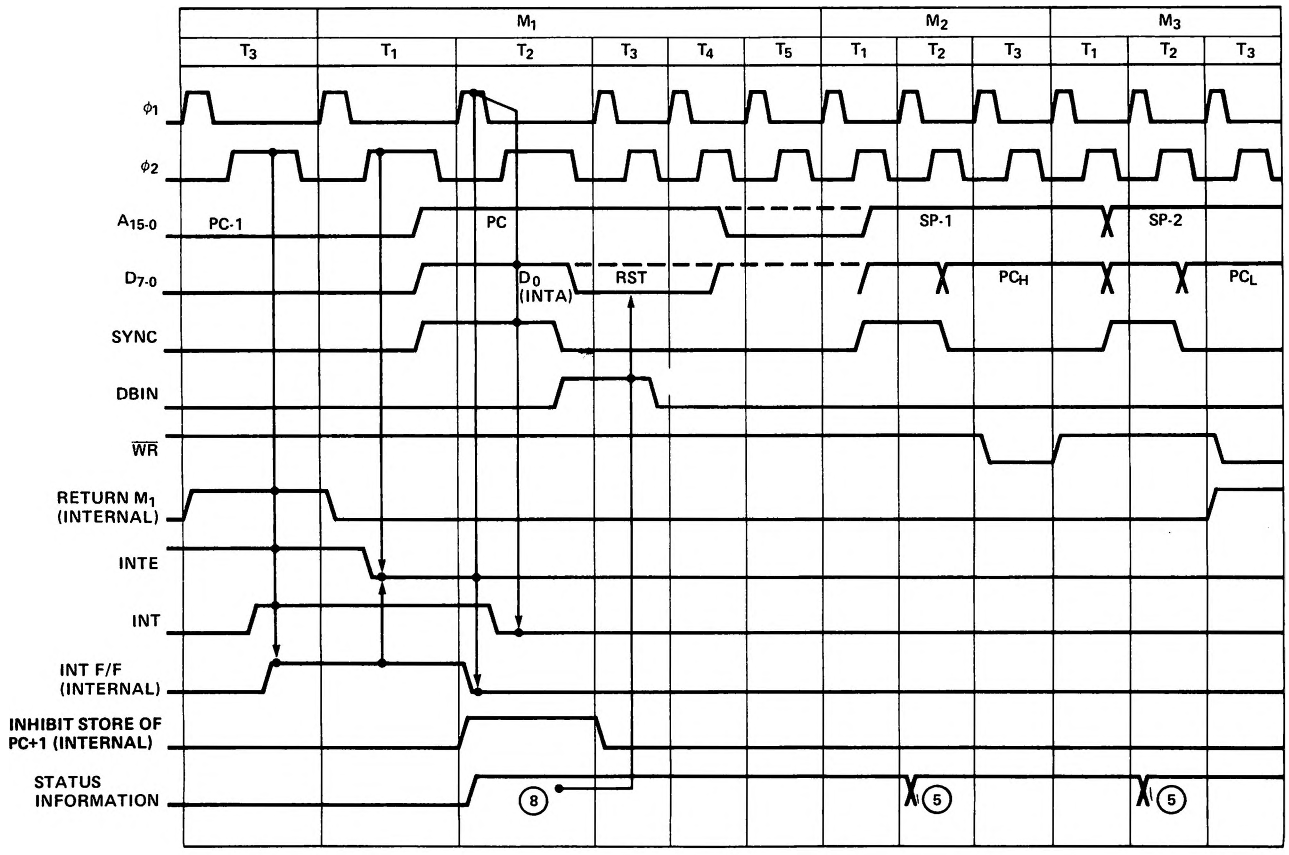
The 8080 has the built-in capacity to handle external interrupt requests. A peripheral device can initiate an interrupt simply by driving the processor's interrupt (INT) line high.

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 ϕ_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: Ⓝ Refer to Status Word Chart on Page 2-6.

Figure 2-8. Interrupt Timing

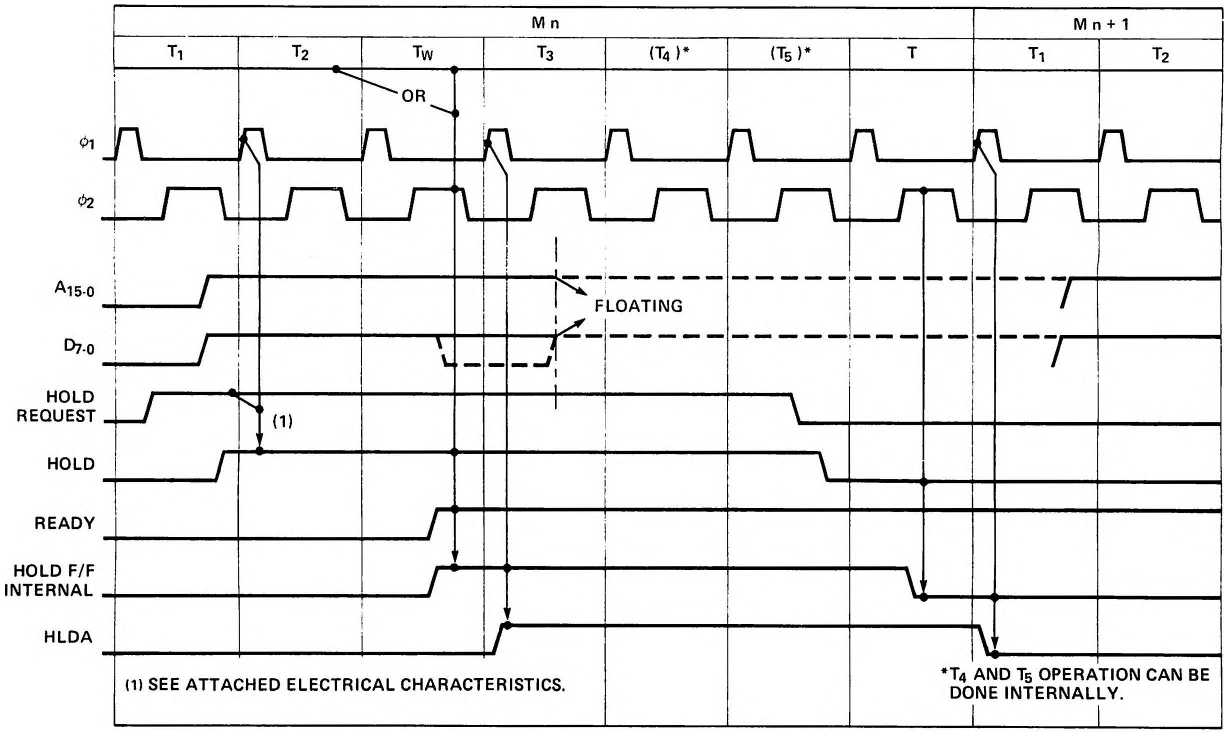


Figure 2-9. HOLD Operation (Read Mode)

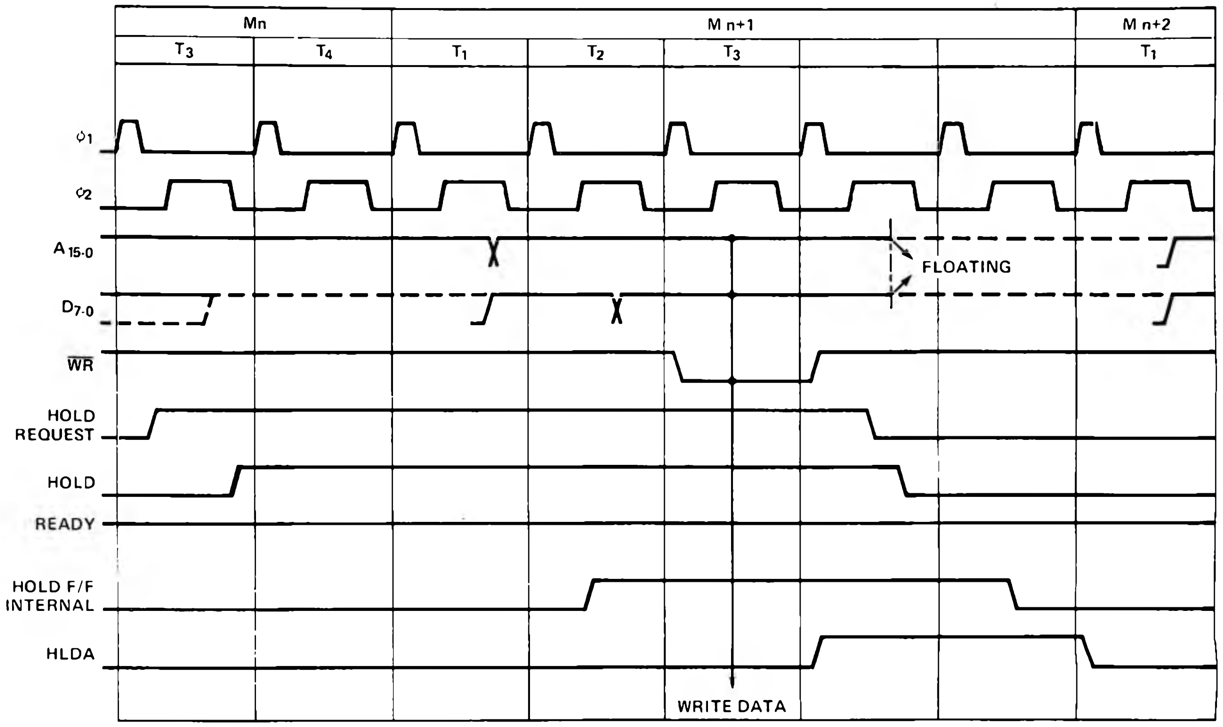


Figure 2-10. HOLD Operation (Write Mode)

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 outpin 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 ϕ_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 ϕ_2 clocks sets the internal hold latch. Setting the latch enables the subsequent rising edge of the ϕ_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 ϕ_1 clock pulse. Address and data lines are floated within a brief delay after the rising edge of the next ϕ_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 ϕ_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 ϕ_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.

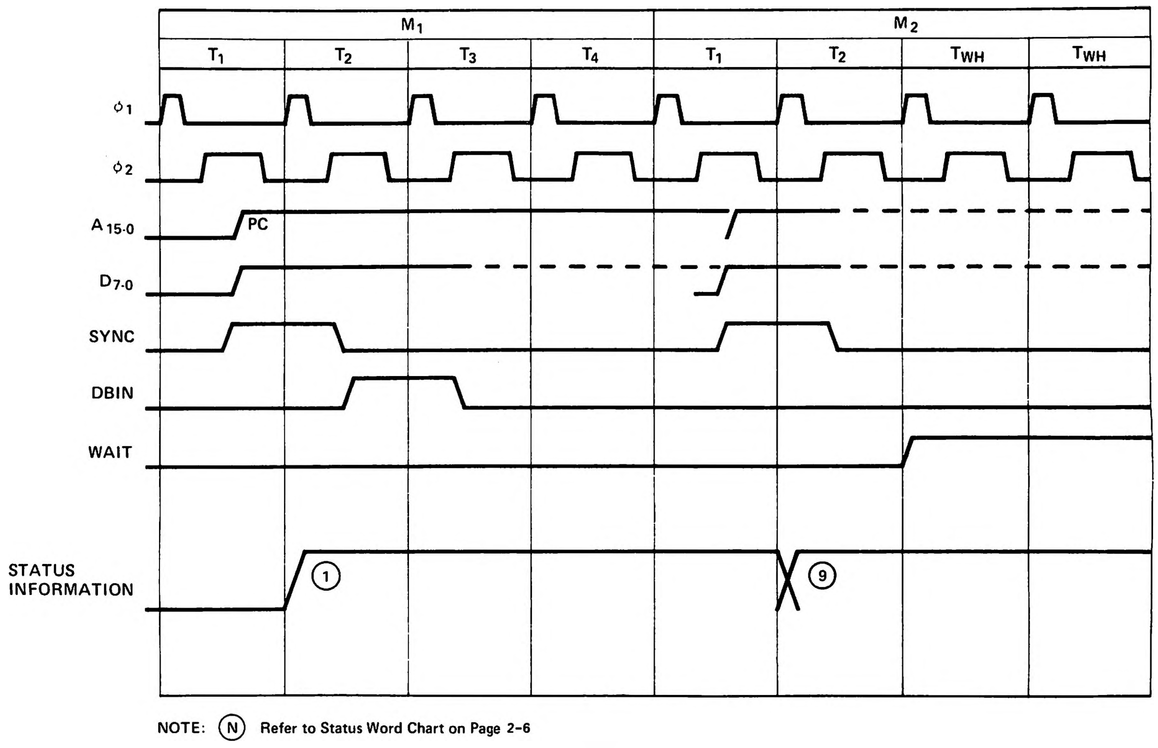


Figure 2-11. HALT Timing

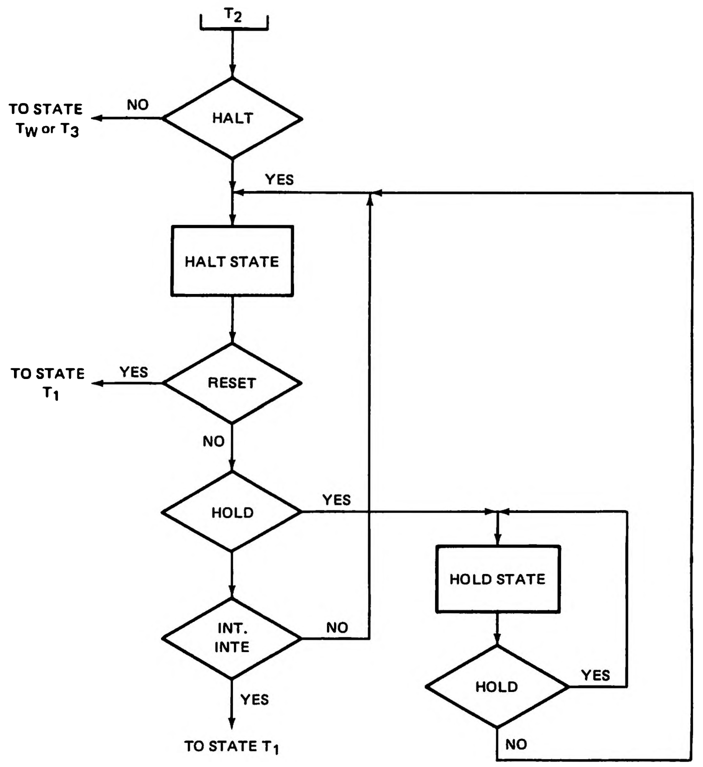
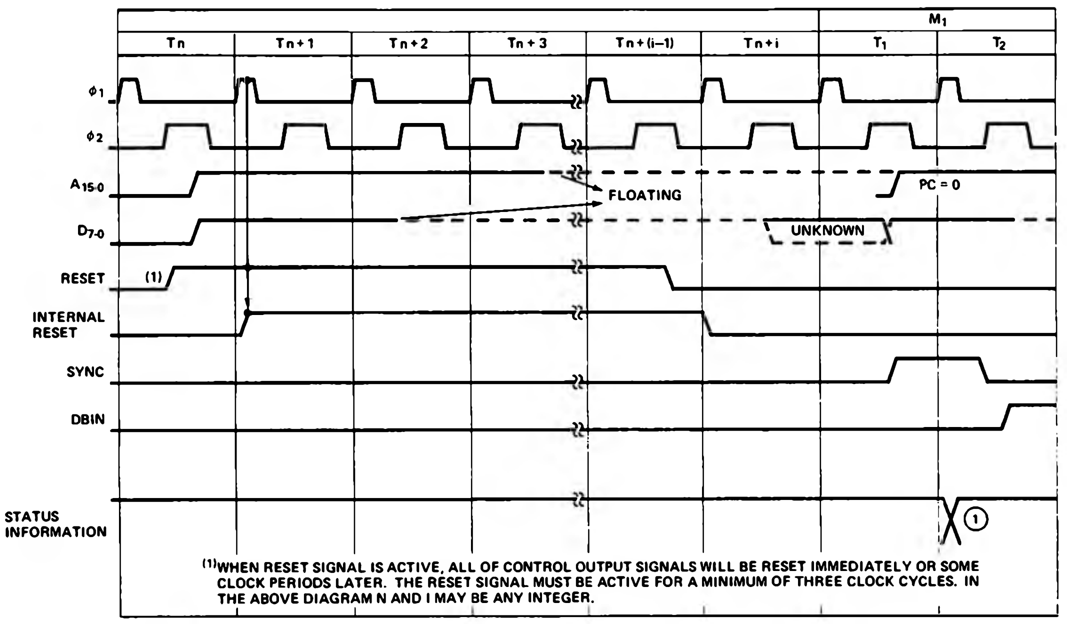


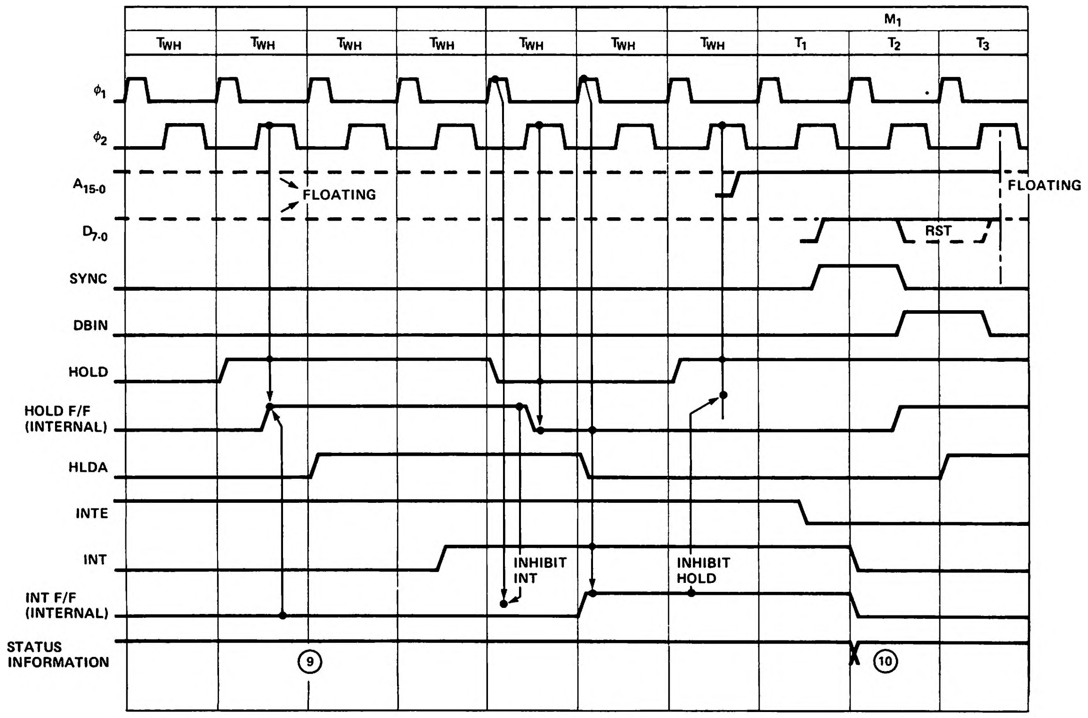
Figure 2-12. HALT Sequence Flow Chart.



(1)WHEN RESET SIGNAL IS ACTIVE, ALL OF CONTROL OUTPUT SIGNALS WILL BE RESET IMMEDIATELY OR SOME CLOCK PERIODS LATER. THE RESET SIGNAL MUST BE ACTIVE FOR A MINIMUM OF THREE CLOCK CYCLES. IN THE ABOVE DIAGRAM N AND I MAY BE ANY INTEGER.

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		
	$D_7 D_6 D_5 D_4$	$D_3 D_2 D_1 D_0$	T1	T2[2]	T3	T4	T5	T1	T2[2]	T3
MOV r1, r2	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		(ri)→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 → WZ = WZ + 1	L	WZ OUT STATUS[6]	DATA →	H		
PC OUT STATUS[6]	PC = PC + 1 B3 →	W	WZ OUT STATUS[7]	(L) → WZ = WZ + 1	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[1]					M2		
	$D_7 D_6 D_5 D_4$	$D_3 D_2 D_1 D_0$	T1	T2[2]	T3	T4	T5	T1	T2[2]	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[6]	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[6]	DATA —	→TMP
XRI data	1 1 1 0	1 1 1 0				(A)→ACT		PC OUT STATUS[6]	PC = PC + 1 B2 —	→TMP
ORA r	1 0 1 1	0 S S S				(A)→ACT (SSS)→TMP		[9]	(ACT)+(TMP)→A	
ORA M	1 0 1 1	0 1 1 0				(A)→ACT		HL OUT STATUS[6]	DATA —	→TMP
ORI data	1 1 1 1	0 1 1 0				(A)→ACT		PC OUT STATUS[6]	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[6]	DATA —	→TMP
CPI data	1 1 1 1	1 1 1 0				(A)→ACT		PC OUT STATUS[6]	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				$\overline{(A)}$→A				
CMC	0 0 1 1	1 1 1 1				$\overline{CY}$→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[6]	PC = PC + 1 B2 —	→Z
J cond addr[17]	1 1 C C	C 0 1 0				JUDGE CONDITION		PC OUT STATUS[6]	PC = PC + 1 B2 —	→Z
CALL addr	1 1 0 0	1 1 0 1				SP = SP − 1		PC OUT STATUS[6]	PC = PC + 1 B2 —	→Z
C cond addr[17]	1 1 C C	C 1 0 0				JUDGE CONDITION IF TRUE, SP = SP − 1		PC OUT STATUS[6]	PC = PC + 1 B2 —	→Z
RET	1 1 0 0	1 0 0 1				X		SP OUT STATUS[15]	SP = SP + 1 DATA —	→Z
R cond addr[17]	1 1 C C	C 0 0 0			INST→TMP/IR	JUDGE CONDITION[14]		SP OUT STATUS[15]	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[16]	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[16]	SP = SP − 1 (rh) —	→DATA BUS
PUSH PSW	1 1 1 1	0 1 0 1				SP = SP − 1		SP OUT STATUS[16]	SP = SP − 1 (A) —	→DATA BUS
POP rp	1 1 R P	0 0 0 1				X		SP OUT STATUS[15]	SP = SP + 1 DATA —	→r1
POP PSW	1 1 1 1	0 0 0 1				X		SP OUT STATUS[15]	SP = SP + 1 DATA —	→FLAGS
XTHL	1 1 1 0	0 0 1 1				X		SP OUT STATUS[15]	SP = SP + 1 DATA —	→Z
IN port	1 1 0 1	1 0 1 1				X		PC OUT STATUS[6]	PC = PC + 1 B2 —	→Z, W
OUT port	1 1 0 1	0 0 1 1				X		PC OUT STATUS[6]	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[20]	
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) → SP = SP - 1	DATA BUS	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) → SP = SP - 1	DATA BUS	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) → (PCL) →	Z DATA BUS									WZ OUT STATUS[11]	(WZ) + 1 → PC
SP OUT STATUS[16]	(rl) →	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_{0-15}) 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_{0-7}) and 8-15 (A_{8-15}).

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.
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

*"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.

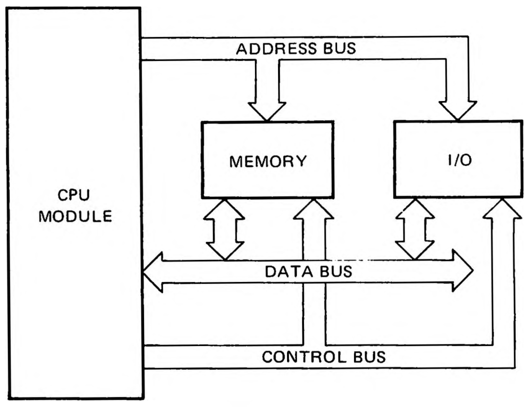


Figure 3-1. Typical Computer System Block Diagram

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,

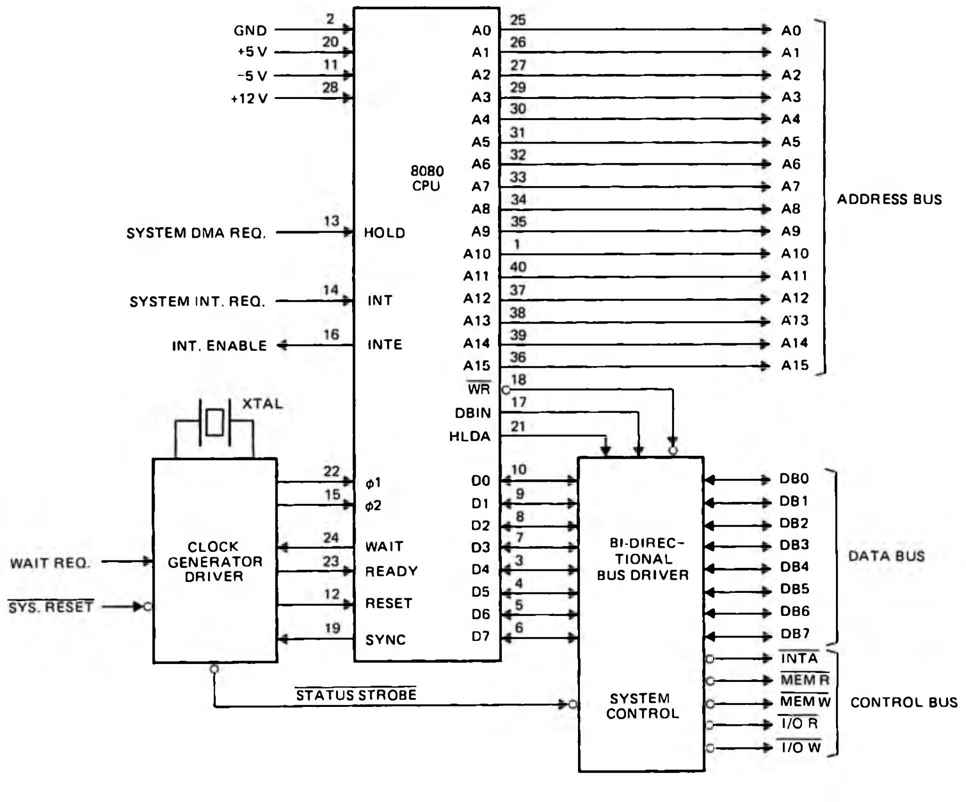


Figure 3-2. 8080 CPU Interface

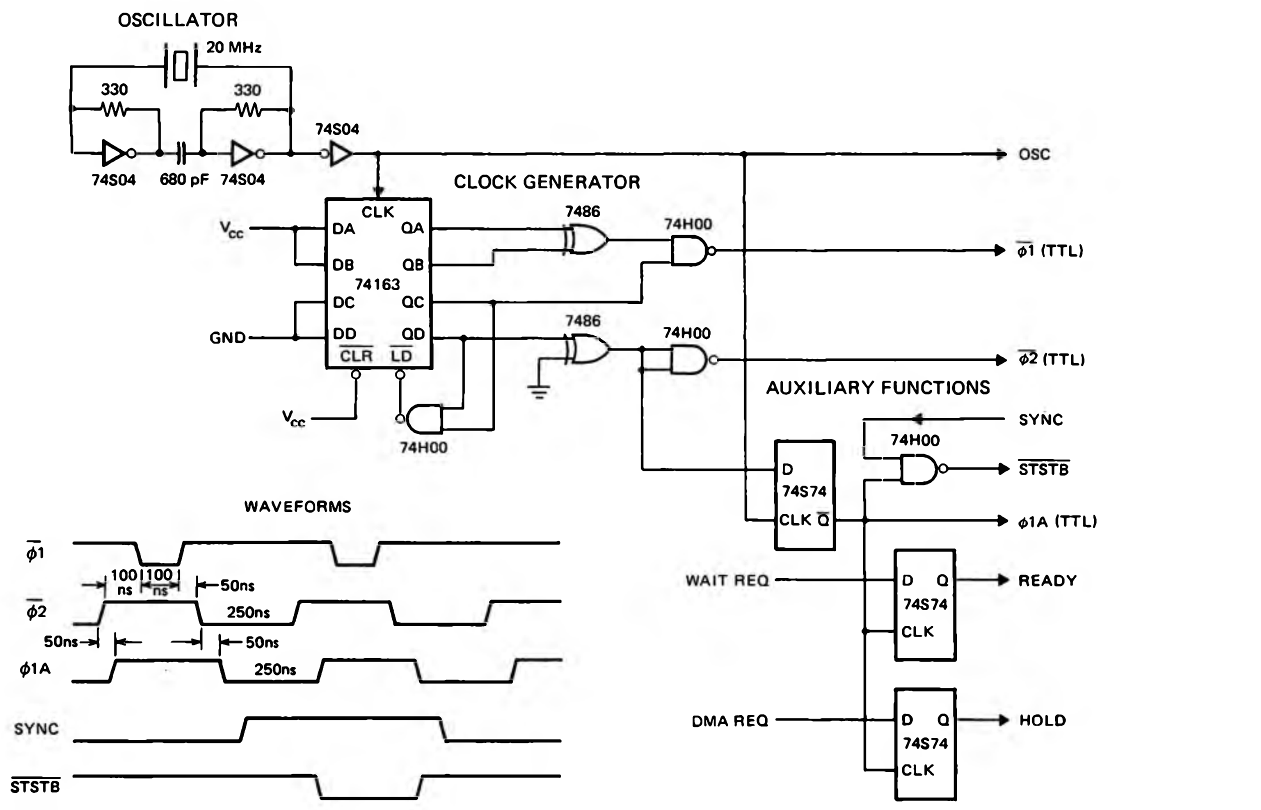


Figure 3-3. 8080 Clock Generator

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, presettable, 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 capacitivity 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.

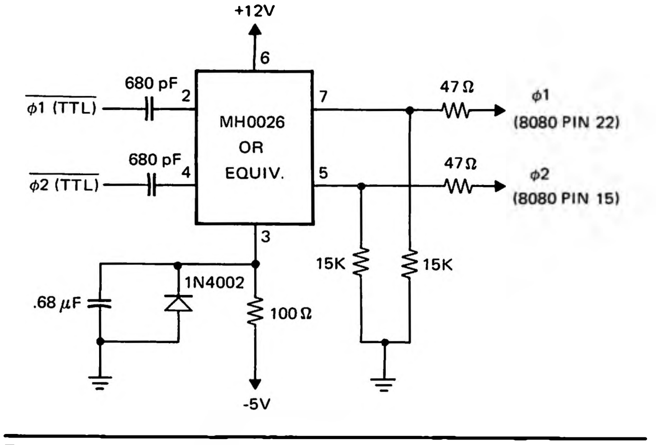


Figure 3-4. High Level Driver

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 (ϕ1A) that is handy to use in clocking "D" type flipflops to synchronize external requests. It can also be used to generate a strobe ($\overline{\text{STSTB}}$) 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 (ϕ1A) 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.

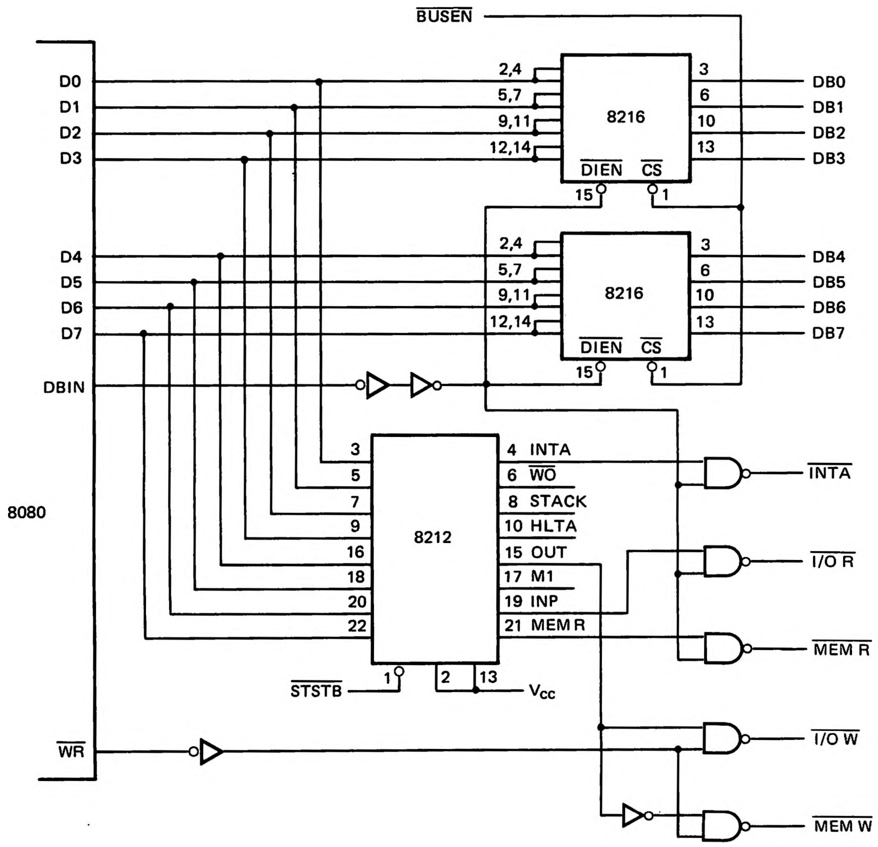


Figure 3-5. 8080 System Control

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® 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 ($\overline{\text{DIEN}}$) so the correct flow of data on the bus is maintained. The chip select ($\overline{\text{CS}}$) of the 8216 is connected to BUS ENABLE ($\overline{\text{BUSEN}}$) 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® 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 ($\overline{\text{STSTB}}$) 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 inDMA dependent environments.

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

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

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

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

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

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

A fifth signal, Interrupt Acknowledge ($\overline{\text{INTA}}$) 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 $\overline{\text{WO}}$, 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® 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 ($\overline{\text{BUSEN}}$) 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.

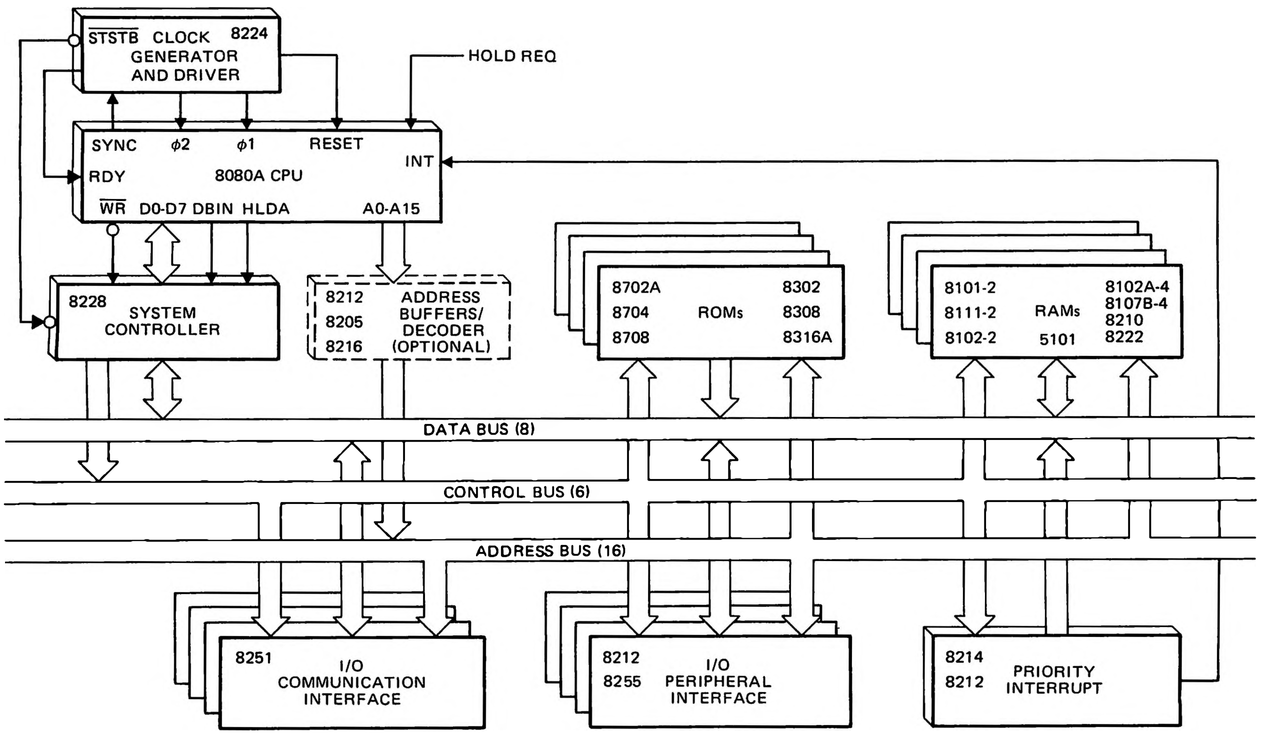
ROM INTERFACE

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



STSTB CLOCK 8224 GENERATOR AND DRIVER
HOLD REQ
SYNC φ2 φ1 RESET
INT
RDY 8080A CPU
WR D0-D7 DBIN HLDA A0-A15
8228 SYSTEM CONTROLLER
8212 8205 8216 ADDRESS BUFFERS/ DECODER (OPTIONAL)
8702A 8704 8708 ROMs 8302 8308 8316A
8101-2 8111-2 8102-2 RAMs 5101 8102A-4 8107B-4 8210 8222
DATA BUS (8)
CONTROL BUS (6)
ADDRESS BUS (16)
8251 I/O COMMUNICATION INTERFACE
8212 8255 I/O PERIPHERAL INTERFACE
8214 8212 PRIORITY INTERRUPT

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® 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® 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.

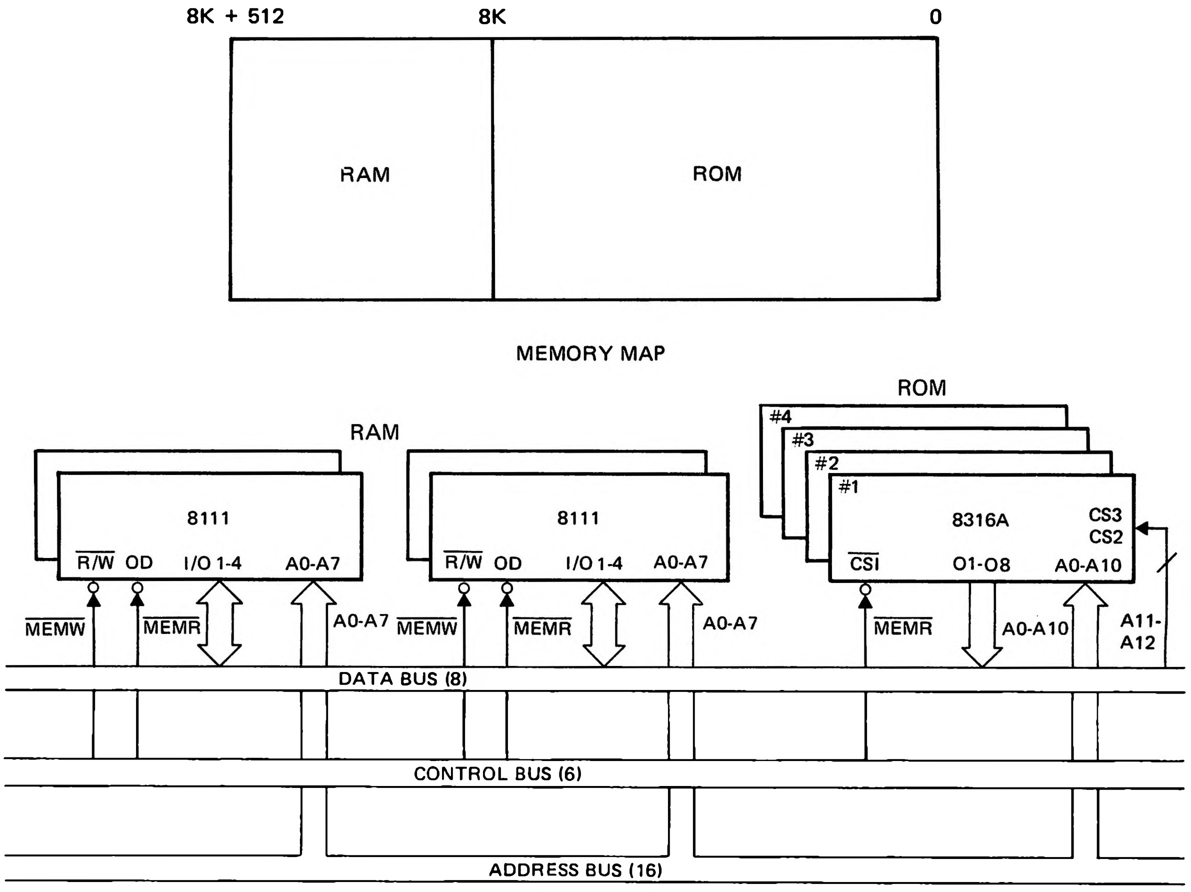


Figure 3-7. Typical Memory Interface

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.

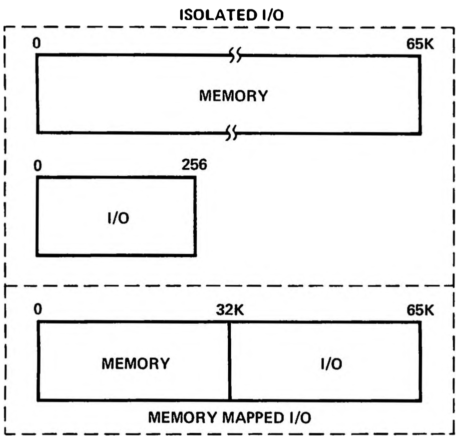


Figure 3-8. Memory/I/O Mapping.

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 uneffected by I/O addressing.

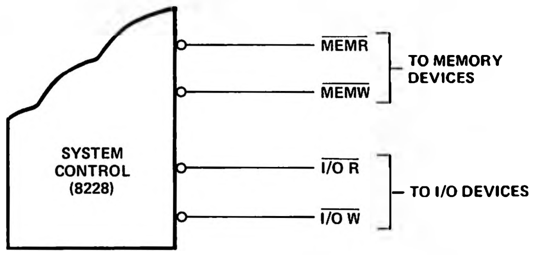


Figure 3-9. Isolated I/O.

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 $\overline{MEMR}$ and $\overline{MEMW}$ signals with A_{15}, 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_{15} as the I/O "flag", a simple method of I/O discipline is maintained:

If A_{15} is a "zero" then Memory is active.
If A_{15} is a "one" then I/O is active.

Other address bits can also be used for this function. A_{15} 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:

MOVr, 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.

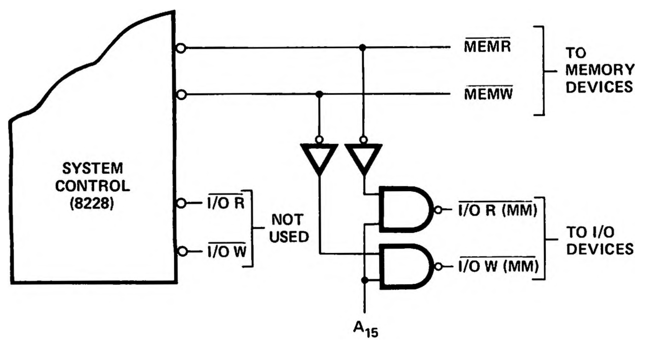


Figure 3-10. Memory Mapped I/O.

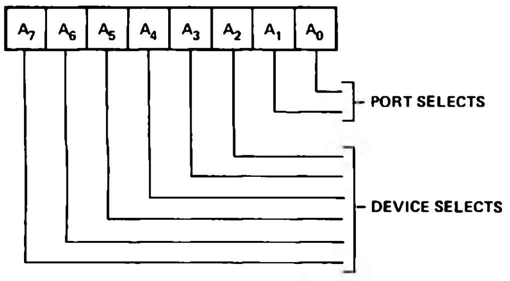
I/O Addressing

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



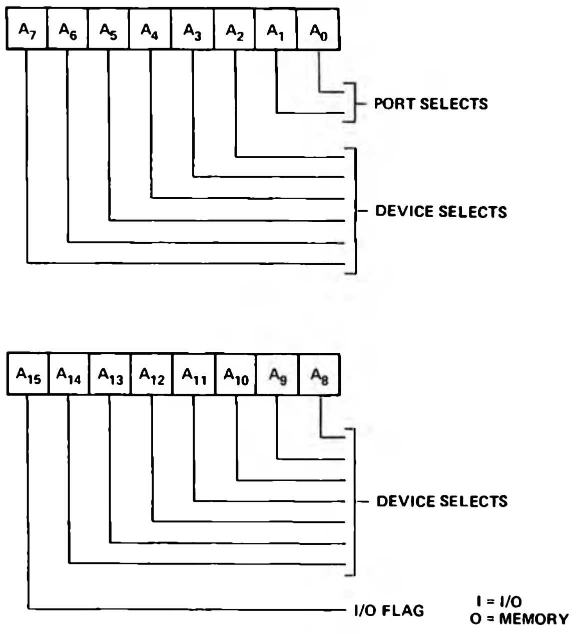
ADDRESSES – 6 – 8255s
(18 PORTS – 144 BITS)

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



ADDRESSES – 13 – 8255s
(39 PORTS – 312 BITS)

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.

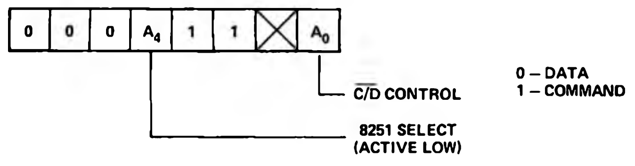


Figure 3-13. 8251 Format.

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.

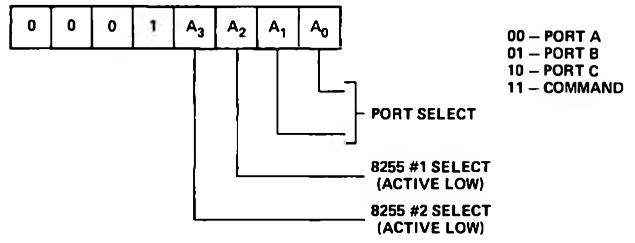
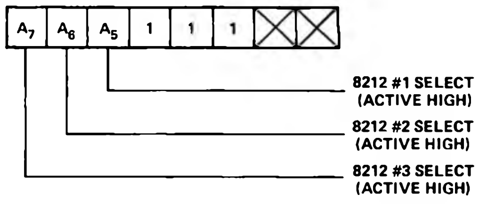


Figure 3-14. 8255 Format.

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



A7
A6
A5
1
1
1
8212 #1 SELECT (ACTIVE HIGH)
8212 #2 SELECT (ACTIVE HIGH)
8212 #3 SELECT (ACTIVE HIGH)

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.

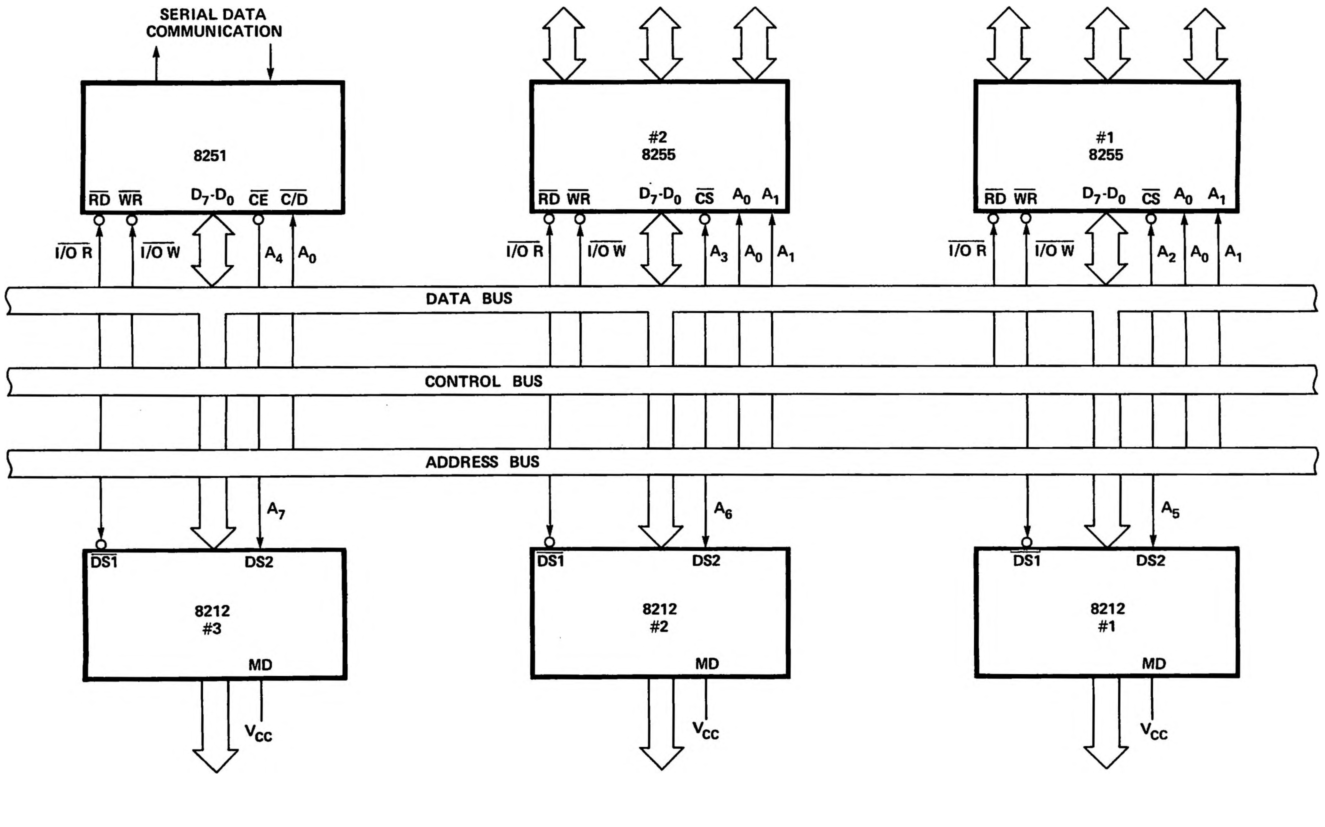


Figure 3-16. Typical I/O Interface.

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:

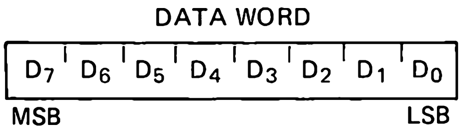
- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group** — add, subtract, increment or decrement data in registers or in memory
- **Logical Group** — AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group** — 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.

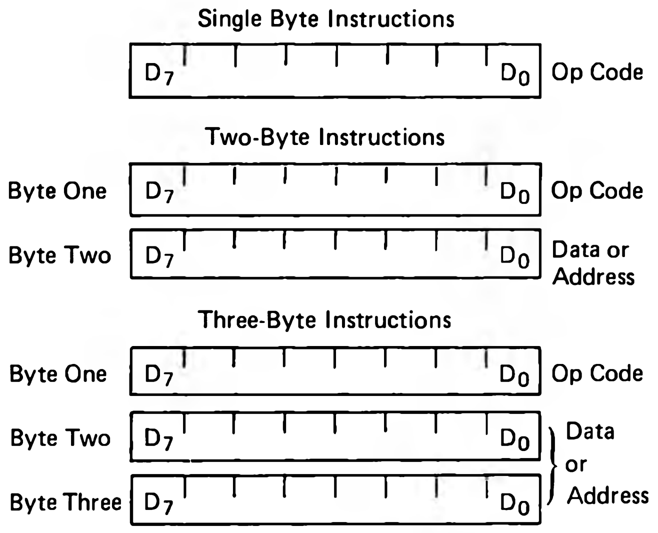
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

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_m	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.
$\leftarrow$	"Is transferred to"
$\wedge$	Logical AND
$\forall$	Exclusive OR
$\vee$	Inclusive OR
+	Addition
−	Two's complement subtraction
*	Multiplication
$\leftrightarrow$	"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 narative 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.

0	1	D	D	D	S	S	S

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.

0	1	D	D	D	1	1	0

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.

0	1	1	1	0	S	S	S

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.

0	0	D	D	D	1	1	0
data							

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.

0	0	1	1	0	1	1	0
data							

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.

0	0	R	P	0	0	0	1
low-order data							
high-order data							

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

LDA addr (Load Accumulator direct)

(A) ← ((byte 3)(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.

0	0	1	1	1	0	1	0
low-order addr							
high-order addr							

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

STA addr (Store Accumulator direct)

((byte 3)(byte 2)) ← (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.

0	0	1	1	0	0	1	0
low-order addr							
high-order addr							

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

LHLD addr (Load H and L direct)

(L) ← ((byte 3)(byte 2))

(H) ← ((byte 3)(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.

0	0	1	0	1	0	1	0
low-order addr							
high-order addr							

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

SHLD addr (Store H and L direct)

((byte 3)(byte 2)) ← (L)

((byte 3)(byte 2) + 1) ← (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.

0	0	1	0	0	0	1	0
low-order addr							
high-order addr							

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

LDAX rp (Load accumulator indirect)

(A) ← ((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.

0	0	R	P	1	0	1	0

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

STAX rp (Store accumulator indirect)

((rp)) ← (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.

0	0	R	P	0	0	1	0

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

XCHG (Exchange H and L with D and E)

(H) ↔ (D)

(L) ↔ (E)

The contents of registers H and L are exchanged with the contents of registers D and E.

1	1	1	0	1	0	1	1

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) ← (A) + (r)

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

1	0	0	0	0	S	S	S

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

ADD M (Add memory)

(A) ← (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.

1	0	0	0	0	1	1	0

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

ADI data (Add immediate)

(A) ← (A) + (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.

1	1	0	0	0	1	1	0
data							

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

ADC r (Add Register with carry)

(A) ← (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.

1	0	0	0	1	S	S	S

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

ADC M (Add memory with carry)

(A) ← (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.

1	0	0	0	1	1	1	0

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

ACI data (Add immediate with carry)

(A) ← (A) + (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.

1	1	0	0	1	1	1	0
data							

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

SUB r (Subtract Register)

(A) ← (A) – (r)

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

1	0	0	1	0	S	S	S

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

SUB M (Subtract memory)

(A) ← (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.

1	0	0	1	0	1	1	0

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

SUI data (Subtract immediate)

(A) ← (A) – (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.

1	1	0	1	0	1	1	0
data							

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

SBB r (Subtract Register with borrow)

(A) ← (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.

1	0	0	1	1	S	S	S

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

SBB M (Subtract memory with borrow)

(A) ← (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.

1	0	0	1	1	1	1	0

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

SBI data (Subtract immediate with borrow)

(A) ← (A) – (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.

1	1	0	1	1	1	1	0
data							

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

INR r (Increment Register)

(r) ← (r) + 1

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

0	0	D	D	D	1	0	0

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

INR M (Increment memory)

((H) (L)) ← ((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.

0	0	1	1	0	1	0	0

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

DCR r (Decrement Register)

(r) ← (r) – 1

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

0	0	D	D	D	1	0	1

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

DCR M (Decrement memory)

((H) (L)) ⟵ ((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.

0	0	1	1	0	1	0	1

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

INX rp (Increment register pair)

(rh) (rl) ⟵ (rh) (rl) + 1

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

0	0	R	P	0	0	1	1

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

DCX rp (Decrement register pair)

(rh) (rl) ⟵ (rh) (rl) – 1

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

0	0	R	P	1	0	1	1

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

DAD rp (Add register pair to H and L)

(H) (L) ⟵ (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.

0	0	R	P	1	0	0	1

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.

0	0	1	0	0	1	1	1

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) ⟵ (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.**

1	0	1	0	0	S	S	S

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

ANA M (AND memory)

(A) ⟵ (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.**

1	0	1	0	0	1	1	0

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

ANI data (AND immediate)

(A) ⟵ (A) ∧ (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.**

1	1	1	0	0	1	1	0
data							

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

XRA r (Exclusive OR Register)

(A) ⟵ (A) ∀ (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.**

1	0	1	0	1	S	S	S

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

XRA M (Exclusive OR Memory)

(A) ⟵ (A) ∀ ((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.**

1	0	1	0	1	1	1	0

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

XRI data (Exclusive OR immediate)

(A) ⟵ (A) ∀ (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.**

1	1	1	0	1	1	1	0
data							

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

ORA r (OR Register)

(A) ⟵ (A) V (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.**

1	0	1	1	0	S	S	S

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

ORA M (OR memory)

(A) ⟵ (A) V ((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.**

1	0	1	1	0	1	1	0

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

ORI data (OR Immediate)

(A) ⟵ (A) V (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.**

1	1	1	1	0	1	1	0
data							

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).**

1	0	1	1	1	S	S	S

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 $(A) < ((H)\ (L))$.

1	0	1	1	1	1	1	0

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

CPI data (Compare immediate)

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

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 $(A) = (\text{byte } 2)$. The CY flag is set to 1 if $(A) < (\text{byte } 2)$.

1	1	1	1	1	1	1	0
data							

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

RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n)\ ;\ (A_0) \leftarrow (A_7)$

$(CY) \leftarrow (A_7)$

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.**

0	0	0	0	0	1	1	1

Cycles:	1
States:	4
Flags:	CY

RRC (Rotate right)

$(A_n) \leftarrow (A_{n-1})\ ;\ (A_7) \leftarrow (A_0)$

$(CY) \leftarrow (A_0)$

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.**

0	0	0	0	1	1	1	1

Cycles:	1
States:	4
Flags:	CY

RAL (Rotate left through carry)

$(A_{n+1}) \leftarrow (A_n)\ ;\ (CY) \leftarrow (A_7)$

$(A_0) \leftarrow (CY)$

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.**

0	0	0	1	0	1	1	1

Cycles:	1
States:	4
Flags:	CY

RAR (Rotate right through carry)

$(A_n) \leftarrow (A_{n+1})\ ;\ (CY) \leftarrow (A_0)$

$(A_7) \leftarrow (CY)$

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.**

0	0	0	1	1	1	1	1

Cycles:	1
States:	4
Flags:	CY

CMA (Complement accumulator)

$(A) \leftarrow (\overline{A})$

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

0	0	1	0	1	1	1	1

Cycles:	1
States:	4
Flags:	none

CMC (Complement carry)

(CY) ← ($\overline{\text{CY}}$)

The CY flag is complemented. **No other flags are affected.**

0	0	1	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

STC (Set carry)

(CY) ← 1

The CY flag is set to 1. **No other flags are affected.**

0	0	1	1	0	1	1	1

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

1	1	0	0	0	0	1	1
low-order addr							
high-order addr							

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.

1	1	C	C	C	0	1	0
low-order addr							
high-order addr							

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.

1	1	0	0	1	1	0	1
low-order addr							
high-order addr							

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

Ccondition addr (Condition call)

If (CCC),
((SP) – 1) ← (PCH)
((SP) – 2) ← (PCL)
(SP) ← (SP) – 2
(PC) ← (byte 3) (byte 2)

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

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

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

RET (Return)

(PCL) ← ((SP));
(PCH) ← ((SP) + 1);
(SP) ← (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.

1	1	0	0	1	0	0	1

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

Rcondition (Conditional return)

If (CCC),
(PCL) ← ((SP))
(PCH) ← ((SP) + 1)
(SP) ← (SP) + 2

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

1	1	C	C	C	0	0	0

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

RST n (Restart)

((SP) – 1) ← (PCH)
((SP) – 2) ← (PCL)
(SP) ← (SP) – 2
(PC) ← 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.

1	1	N	N	N	1	1	1

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

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

Program Counter After Restart

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

(PCH) ← (H)
(PCL) ← (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.

1	1	1	0	1	0	0	1

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

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

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.**

1	1	R	P	0	1	0	1

Cycles:	3
States:	11
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.

1	1	1	1	0	1	0	1

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.**

1	1	R	P	0	0	0	1

Cycles:	3
States:	10
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.

1	1	1	1	0	0	0	1

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

XTHL (Exchange stack top with H and L)

(L) ⟷ ((SP))
(H) ⟷ ((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.

1	1	1	0	0	0	1	1

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

SPHL (Move HL to SP)

(SP) ⟵ (H) (L)

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

1	1	1	1	1	0	0	1

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

IN port (Input)

(A) ⟵ (data)

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

1	1	0	1	1	0	1	1
port							

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

OUT port (Output)

(data) ⟵ (A)

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

1	1	0	1	0	0	1	1
port							

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

EI (Enable interrupts)

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

1	1	1	1	1	0	1	1

Cycles:	1
States:	4
Flags:	none

DI (Disable interrupts)

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

1	1	1	1	0	0	1	1

Cycles:	1
States:	4
Flags:	none

HLT (Halt)

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

0	1	1	1	0	1	1	0

Cycles:	1
States:	7
Flags:	none

NOP (No op)

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

0	0	0	0	0	0	0	0

Cycles:	1
States:	4
Flags:	none

INSTRUCTION SET

Summary of Processor Instructions

		Instruction Code[1]								Clock[2]
Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Cycles
MOV$_{r1,r2}$	Move register to register	0	1	D	D	D	S	S	S	5
MOV M, r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r, M	Move memory to register	0	1	D	D	D	1	1	0	7
HLT	Halt	0	1	1	1	0	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
INR r	Increment register	0	0	D	D	D	1	0	0	5
OCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
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

		Instruction Code[1]								Clock[2]
Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Cycles
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	C	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RST	Restart	1	1	A	A	A	1	1	1	11
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4

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.