Lecture-13

Internal Architecture of Intel 8085A

The functional block diagram of 8085A is shown in fig.4.1.

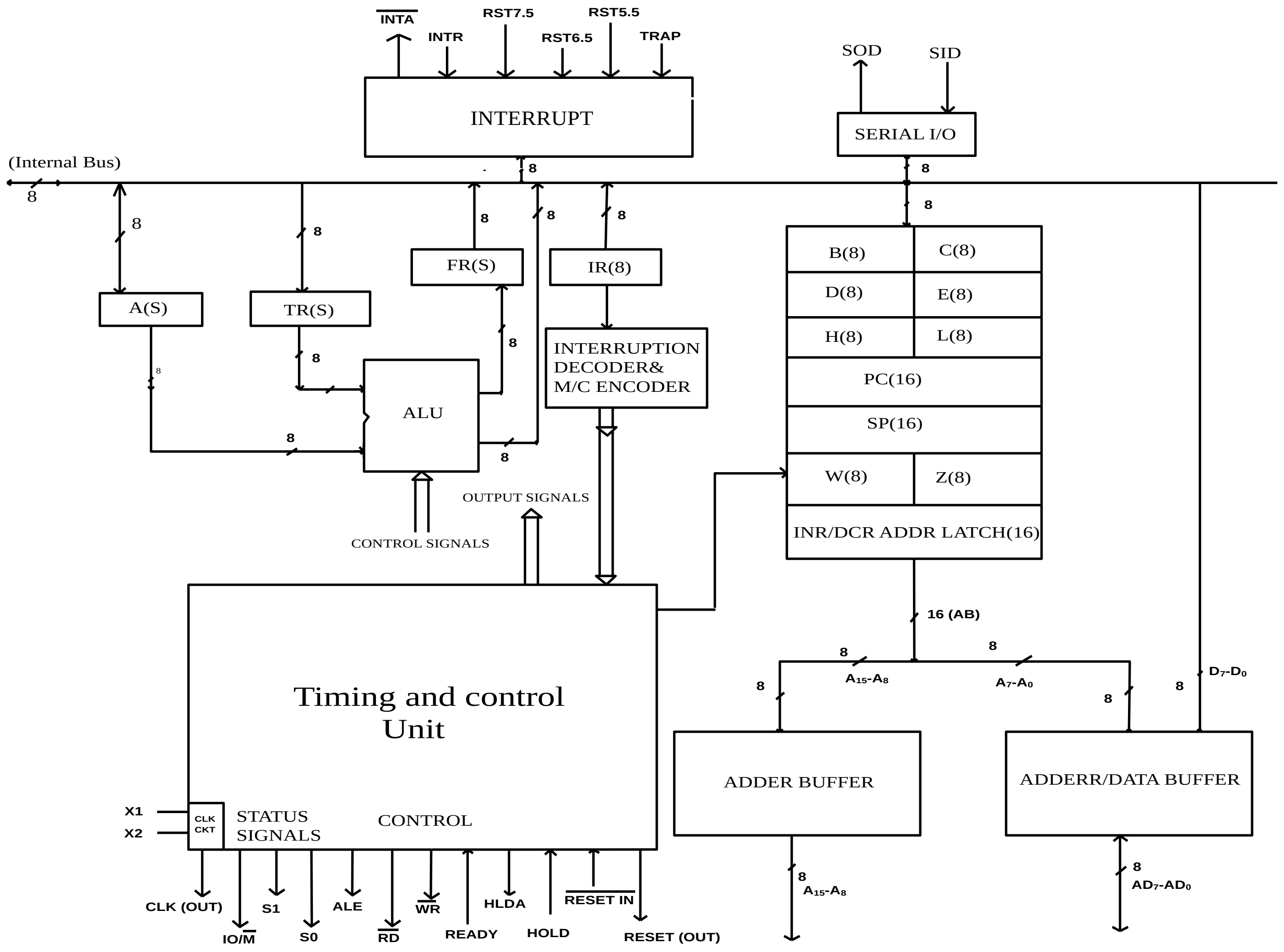


Fig.4.1 Internal Architecture of 8085A Microprocessor

It consists of five essential blocks.

(1) Arithmetic Logic Section

(2) Register Section

(3) The Interrupt Control Section

(4) Serial I/O Section

(5) The Timing And Control Unit

There is an internal bi-directional data bus of 8-bit wide. This bus is used to transfer data and instructions among various internal registers. All the internal registers which transfer data to the internal bus are tri-state registers. Higher order address bus (A_{15}-A_8) and time-multiplexed lower order address data bus (AD_7-AD_0) are the external buses and used to interface peripherals and memory chips to CPU. Address buffer and address/data buffers isolate the internal data bus from the external address bus and address/data bus and drive the external address bus and address/data bus. The CPU can send the address of desired memory locations and I/O chip through these buffers. The 8-bit internal data bus is also connected to the address/data buffers. The bi-directional arrows indicate a tri-state connection that allows the address/data buffer to send or receive data from the 8-bit internal data bus. In the output mode the information on the data bus is loaded into the 8-bit data latch that drives the address/data bus output buffer. The output buffers are floated during input or non transfer operations. During the input mode, data from the external bus is transferred over the internal data bus to internal register.

The figure shown does not include the control signals driving internal registers.

Arithmetic & Logic Section: This section consists of:

(a) Accumulator (A)
(b) Temporary Register (TR)
(c) Flag Register (FR)
(d) Arithmetic Logic Unit (ALU)

Accumulator:

Arithmetic and/or logic operations on one or two operations are the basic data transformations implemented in a μρ one of these two operands is always in the accumulator. Accumulator is an 8-bit register accessible to the user is connected to the 8-bit internal data bus. The bi-directional arrow between the accumulator and the bus indicates a tri-state connection that allows the accumulator to send or receive data. In addition, it has a two state 8-bit output. The content of the accumulator is always available at this two state output as one of the operands for the ALU. The contents of the accumulator can be manipulated through instructions. Its content can be incremented and decremented. The content of the memory location can be transferred to the accumulator and vice-versa. The result of arithmetic/ logical operations carried out by ALU is also stored back in the accumulator. In other words, it accumulates the result of the operation, hence, the name accumulator.

Temporary registrar (TR):

This is an 8-bit register not accessible to the user. It is used by the processor for internal operations. The second operand as and when necessary is loaded in to this register by the microprocessor before the desisted operation takes placed in the ALU. The temporary register has 8-btis two state output. The second operand is always available at this output.

<u>Arithmetic Logical Unit (ALU):</u>

ALU is a combination logic block which performs the desired operation on the two operands. The contents of the accumulator and the temporary register are the inputs to the ALU. This is governed by the control signals generated by the timing and control unit. The various arithmetic and logical operations that can be performed by ALU are:

- Binary addition, subtraction, increment and decrement,
- Logical AND, OR and EX-OR,
- Complement,
- Rotate left of right.

The result of the operation is, in general, stored back in accumulator. In subtraction operation, the content of the temporary register is subtracted from the content of the accumulator and is stored back in the accumulator.

In many applications it is appropriate to represent data in binary coded decimal (BCD) form. The result on any operation on BCD should also be in BCD form. The ALU contains additional logic to adjust result of addition operations where the operands are interpreted as BCD data.

<u>Flags register</u>:

The ALU influences a number of flip flops called flags which store information related to the results of arithmetic and logical operations. Taken together this flags constitute a flag register.

Flag register is an 8-bit register accessible to the user through instruction. Each bit in the flag register has a specific function. Only 5 bits out of 8 bits are used as shown below:

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

The three crossed bit are redundant bits and not used. They can be either ‘0’ or ‘1’ but normally they are forced to be zero. The other five bits are affected as a result of execution of an instruction. All instructions do not affect these flags e.g. data transfer operation do not affect these flags. The meaning and the effect of these flags are as follows.

CY (Carry) Flag bit:

This particular bit is SET (=1) if there is a carry from the MSB position during an addition operation or if there is a borrow during the subtraction operation otherwise the flag is reset (=0). The processor, by design, does the subtraction operation also by taking 2’s complement of one operand and adding it to another operand.

P (Parity) Flag bit:

The parity flag test for the number of ‘1’s in the accumulator. If the accumulator holds on an even number of 1’s, it is said that even parity exists and the parity flag is set to ‘1’. However, if the accumulator holds an odd number of ‘1’ it is called odd parity and the parity flag is reset to ‘0’. In other words, if the module-2 sum of the bit is ‘0’, this flag is set otherwise the flag is reset.

AC (Auxiliary Carry) Flag bit:

This bit is set if there is a carry from b_3 bit to b_4 bit of accumulator during addition operation otherwise it is reset. The AC flag is useful for BCD arithmetic and is used in a particular instruction known as DAA (Decimal Adjust Accumulator).

Z (Zero) Flag bit:

Zero flag bit is SET if the result of an operation is zero, otherwise it is RESET.

Sign Flag bit:

The sign flag is set to the condition of the most significant bit of the accumulator following the execution of arithmetic or logical operation. These instructions use the MSB of the data (result) to represent the sign of the number contained in the accumulator. A set sign flag represents a negative number, where as a reset flag means a positive number.

Example 1:

Let us consider the execution of the instruction ADD B.

ADD is the mnemonic for addition. The first operand is known to exist in the accumulator (Reg. A). Register B contains the second operand. The meaning of the instruction is add the contents of the B register to the contents of A register and store the result back in the accumulator (A). Symbolically we can write,

$$(A) \longleftarrow (A) + (B)$$

Let as suppose the register contents are (A) = 9BH, (B) = A5H before the execution of the instruction. It means,

$$\begin{array}{lcl} (A)=9BH & \rightarrow & (1001\ \ 1011)_2 \\ (B)=A5H & \rightarrow & \underline{(1010\ \ 0101)_2} \\ ADD\ \ B\ =(A+B) & \rightarrow & (0100\ \ 0000)_2 \end{array}$$

As a result of addition, there is a carry from b_3 to b_4 position and therefore AC is set. Also there is a carry from the MSB out and, therefore, CY flag is also set. Soon after the execution of ADD B instruction the accumulator contains (A) = $(0100\ 0000)_2$ = 40H and is non-zero. Therefore Z flag is reset to zero. Also, result contains only one ‘1’, an odd number. Therefore, parity bit is also be reset to zero. Since the MSB of the result is zero, therefore the sign (S) bit is also reset. Thus the flag register, soon after the execution of the instruction, contains $(0001\ 0001)_2 = 11_H$.

Example 2:

Let us consider the execution of another instruction SUB B.

SUB is the mnemonic for subtraction. Accumulator consists of first operand. Register B contains the second operand. The meaning of the instruction is subtract the contents of the B register from the contents of A register and store the result back in the accumulator (A). Symbolically we can write,

$$(A) \longleftarrow (A) - (B)$$

Let as suppose the register contents are (A) = A5H, (B) = 9BH before the execution of the instruction. It means,

Before execution A = A5H and B = 9BH

(A) = 1010 0101 $\rightarrow$ $(1010\ \ 0101)_2$

(B) = 1001 1011 2's complement $\rightarrow$ $\underline{(0110\ \ 0101)_2}$

Carry 1 $\nwarrow$ $(0000\ \ 1010)_2$

Since result is non zero, therefore, Z bit is ‘0’. Sign bit is also ‘0’ because MSB of the result is ‘0’. AC is also ‘0’ because in addition (2’s complement), there is no carry from b_3 to b_4. Parity bit is ‘1’ (2 ones). CY bit seems to be ‘1’. But it is complemented and then stored. Therefore, CY bit is stored as ‘0’. It also indicates that (A) is having larger number than register (B), otherwise smaller one. Thus the flag register, after the execution of the instruction, contains $(0000\ 0100)_2$ = 04H.

Let us consider (A) is having 9B H and (B) is A5 H before execution.

(A) = 1001 1011 $\rightarrow$ $(1001\ \ 1011)_2$

(B) = 1010 0101 2's complement $\rightarrow$ $\underline{(0101\ \ 1011)_2}$

Carry 0 $\nwarrow$ $(1110\ \ 0110)_2$

Therefore, in this case, the flag bits will be S=1, Z=0, AC=1, P=1, CY=1 (complement of ‘1’ obtained in addition). Thus the flag register, after the execution of the instruction, contains $(1001\ 0101)_2$ = 95H.

Let use consider execution of another instruction DCR C. DCR is the mnemonic for decrement register. C register is the operand. This instruction means decrement the content of the C register by ‘1’ and store it back in the C register. The MACRO RTL implemented is

$$(\mathbf{C}) \leftarrow (\mathbf{C}) \mathbf{- 1})$$

Let us suppose (C) contains D2H before the execution of the instruction. After the execution of instruction, (C) shall contains D1H and, therefore, is not zero. Therefore the flag register will contain

$(1000\ 0100)_2$ or 84H. On the other hand, if (C) contains 01H just before the execution of the instruction DCR C. Just after the execution of the instruction, (C) shall contain 00H. Since the result of the operation is ‘0’ the zero flag shall now be SET to ‘1’. Other flag will be affected in the normal way.

These flag bit are utilized in many instructions for branching operations. During the execution of a program normally one of these bits are tested for TRUE & FALSE condition. Depending upon the condition the program branches to different paths. This is shown in fig.4.2

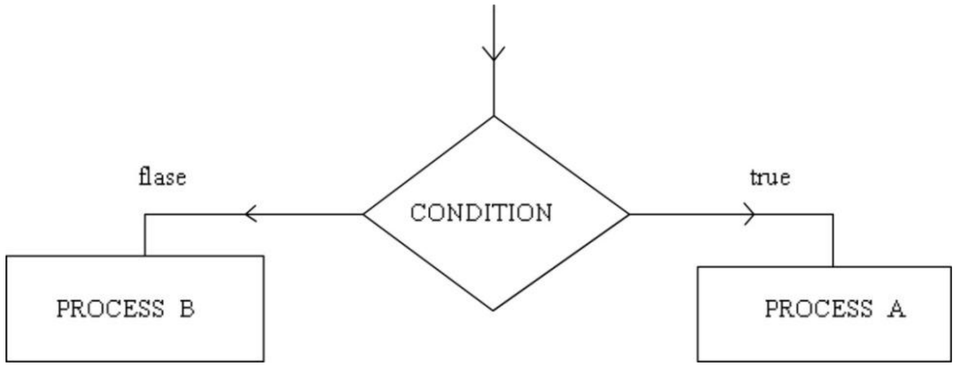


Fig.4.2 Branching Operation Depending on Condition

Lecture-14

REGISTER SECTION:

There are six 8-bit general purpose registers designated as B, C, D, E, H and L. All these registers are accessible to the user. It means their contents can be read without destroying it or some new data can be written into it through instructions. These registers constitute a register array like a small on-chip RAM with addressable memory location. Internal control signals select the register for a read or write operation. This means that the CPU can either load a register from the 8-bit internal data bus or output the register content to the internal 8-bit data bus. Data can also be transferred or exchanged among registers. In an instruction, these six registers along with the accumulator (A) is identified by a 3-bit code designated either SSS or DDD. Whenever SSS is used, it corresponds to source register and whenever DDD is used, it corresponds to destination register. The address codes used for these internal registers are as follows.

SSS or DDD		
000	$\longrightarrow$	(B)
001	$\longrightarrow$	(C)
010	$\longrightarrow$	(D)
011	$\longrightarrow$	(E)
100	$\longrightarrow$	(H)
101	$\longrightarrow$	(L)
110	$\longrightarrow$	(M)
111	$\longrightarrow$	(A)

Note: In the above codes (110) is assigned to memory pointer or M-register. Whenever it is used for SSS or DDD it means a specific register pair (H,L) together forms a 16 bit register known as memory address register (MAR) or M- pointer. In other words, whenever M is used in an instruction, it is assumed the 16-bit address of memory location, being referred, is available in (H,L) register pair.

As an example consider the instruction

$$\text{MOV} \quad r_1, r_2$$

This is an ALP statement. MOV is the mnemonic for move operation. r_1 and r_2 are the operand registers. In this statement r_2 is the source register and r_1 is the destination register. The meaning of the instruction is ‘move the contents of r_2 register into r_1 register. Symbolically this basic operation can be described by a macro RTL statement:

$$(r_1) \longleftarrow (r_2)$$

This is a single byte instruction and this single byte is the operation code. The arrangement of the operation code (op-code) of is shown below:

$$\boxed{0}\boxed{1}\underbrace{\boxed{D}\boxed{D}\boxed{D}}_{r_1\text{ code}}\underbrace{\boxed{S}\boxed{S}\boxed{S}}_{r_2\text{ code}}$$

E.g., the op-code for MOV A, H is $(01\ 111\ 100)_2$ =7CH. When the instruction 7CH is executed the content of (H) register is transferred to (A) register. Note that the content of (H) register is not destroyed. However, the original content of register (A) is lost.

Let us consider another instruction MOV D, M. This is also a single byte instruction. Memory pointed by (H,L) register pair is the source operand and (D) is the destination register.

The meaning of this instruction is move the content of the memory location whose address is available in (H,L) register pair into the (D) register. The opcode of this instruction is $(01\ 010\ 110)_2$ = 56H. Whenever this instruction is executed, the content of the memory location pointed by (H,L) register pair is loaded into the (D) register. The content of the memory location is not destroyed. However, the content of the memory location Y_1Y_0 H whose address is $X_3X_2X_1X_0$ H available in (H,L) pair goes into the D register. The original content of D is lost. This is illustrated in fig.4.3.

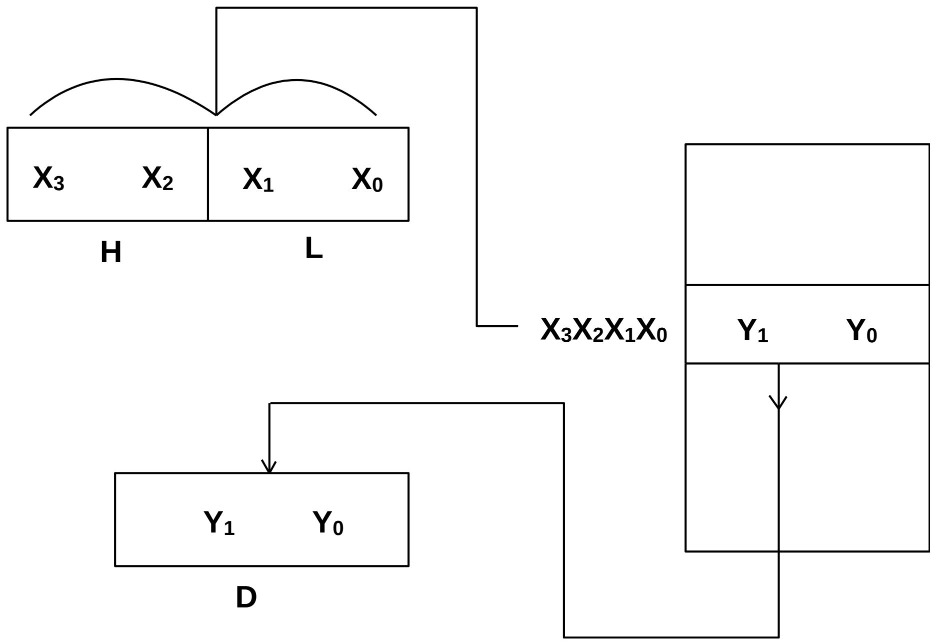


Fig.4.3

The six general purpose registers B, C, D, E, H, L can also be combined together as register pairs for 16-bit operation only the following pairs are possible:

i. (B,C) pair with (C) lower order 8-bits and (B) higher order 8-bits.
ii. (D,E) pair with (E) lower order 8-bits and (D) higher order 8-bits.
iii. (H,L) pair with (L) lower order 8-bits and (H) higher order 8-bits.

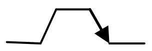
There is another register called stack pointer (SP) which is 16-bit register itself. Whenever an instruction refers to the register pair (B,C), (D,E), (H,L) or (SPH,SPL), an 8-bit code RP is used in the operation code to identify the register pairs.

(RP)		
0 0	$\longrightarrow$	(B,C)
0 1	$\longrightarrow$	(D,E)
1 0	$\longrightarrow$	(H,L)
1 1	$\longrightarrow$	(SPH,SPL)

PROGRAM COUNTER:

This is a 16-bit register accessible to the user. It is a special purpose register and it always contains the address of the next instruction to be fetched from the program memory and executed by the CPU in a program sequence. Thus the program counter keeps the track of the program execution in which instructions are to be executed next.

Whenever necessary in the program execution, the address information available in PC is sent out to the address lines during T_1 timing slot of a machine cycle. The higher order 8-bits of program counter (PCH) are sent out through A_{15}–A_8 address lines & the lower order 8-bits of program counter (PCL) are sent out through AD_7–AD_0 lines during T_1 states. Since the BDB contains the lower order 8-bit address information during T_1 state only, an ALE pulse is also issued by the processor. The above statement can be symbolically stated through macro RTL shown in the figure 22.

T_1: A_{15}-$A_8 \leftarrow$ (PCH), AD_7-$AD_0 \leftarrow$ (PCL), ALE = 

T_2: (PC) $\leftarrow$ (PC) +1

Whenever the address information sent from the program counter to the address bus (external world) during T_1 state, then the (PC) shall be incremented by 1 during the subsequent T_2 state so that program counter points to the next sequential byte. If may be the data required if the previous instruction is of two bytes or three bytes long or it may be the next instruction to be fetched and executed. If instructions are sequentially arranged in memory, this will guarantee that they will also be executed sequentially. Sometimes, program execution requires that non-sequential instructions executed e.g. JMP or CALL type instructions. These instructions require the program counter to be loaded with an entirely new value. An 8-bit microprocessor with a 16-bit program counter requires two data moves to completely modify the contents of the PC.

Note: If the address information for PC has not been sent out during T state to the external world, them the PC will not be incremented using T_2 state.

When the microprocessor is RESET, the CPU initializes the PC to 0000 H. Therefore, the first instruction of the program should be at 0000 H in the memory address space of the CPU.

STACK POINTER REGISTER:

The stack is a storage area of the processor. It consists of number of sequential and RWM locations in which microprocessor saves the internal register contents during subroutine calls and interrupts so that they will not be changed or destroyed by a subroutine.

8085A μp can address directly 64K memory locations. This is known as directly addressable memory space starting from the address 0000H to FFFFH. This entire memory area is usually divided by the user into program area, data area and stack area. It is for the user to see that program area and data area do not overlap with that of stack memory area. The size of the stack memory area depends upon the application.

For example, the user for a particular process control operation may decide to reserve memory space starting from 2600H to 2700H as the stack memory space. This is shown in fig.4.4.

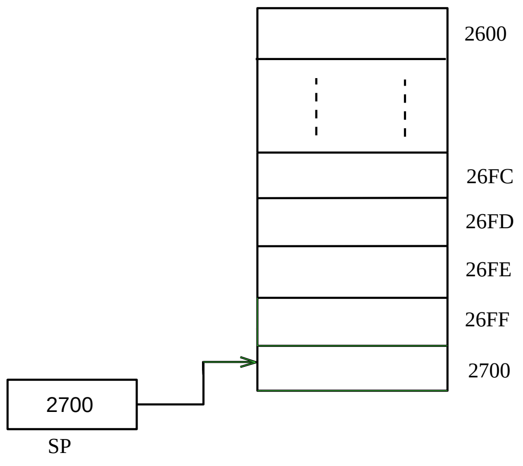


Fig.4.4

The stack pointer is a 16-bit register accessible to the user. It is required to refer any memory location of the stack. It contains the address of the top of stack into which last data is put or written. Writing data into a stack is called a PUSH operation and reading data from a stack is called a POP operation. In the figure shown 2700H is known as the bottom of the stack.

There is an instruction in the instruction set to initialize the stack pointer register to the bottom of the stack. This instruction is

LXI   SP, BADR.

LXI is the mnemonic for Load immediate. BADR is a symbolic name given to the 16-bit address which is to be loaded into the stack pointer. The meaning of the instruction is to load the 16-bit of data immediately available in the instruction itself into the stack pointer. In this example, BADR equals 2700 H. When this instruction is executed the situation is shown in figure. The stack pointer now points to the bottom of the stack.

Now, let us suppose that while calling a subroutine it becomes necessary to save the contents of (B,C) register pair and (D,E) register pair as they are to be used in the subroutine. The process of saving the content of a register is known as push operation. The push operation is performed at the beginning of a subroutine to save register contents and the instruction for pushing the contents of the internal register is PUSH e.g. PUSH B. The meaning of the instruction is to push the contents of (B,C) register pair on to the stack so that it can be saved there till it is restored. PUSH B operation affects the stack and stack pointer as follows:

Since the stack pointer always holds the address of the last byte of data pushed onto the stack, therefore, when PUSH B instruction is executed, the stack pointer is decremented by 1 and the contents of the (B) register are copied onto the stack at that address. The stack pointer is decremented again, and the contents of the (C) register are copied to that address. Just after the execution of the PUSH B instruction, the situation is shown in fig.

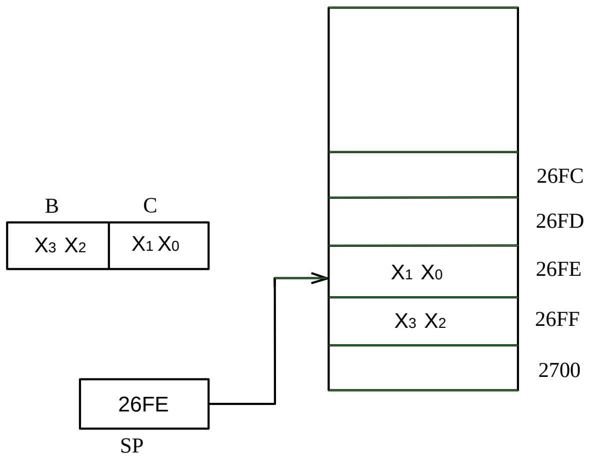


Fig.4.5

Similarly, to store the contents of (D,E) register pair PUSH D instruction is used. The meaning of this instruction is push the contents of the (D,E) pair onto the stack to save them there as shown in figure just after the execution.

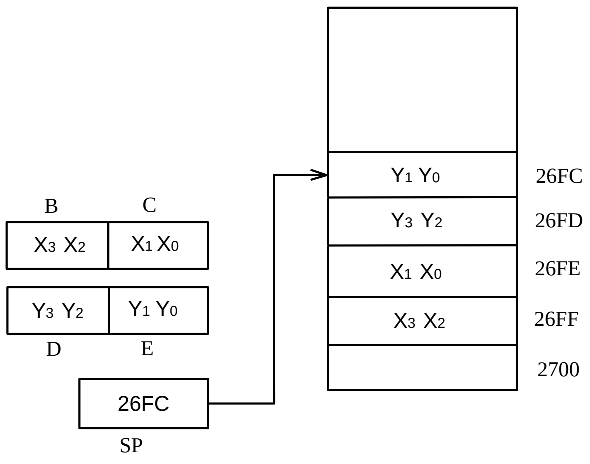


Fig.4.6

Since the contents of (B,C) & (D,E) register pairs are stored at the top of the stack, these registers are now available for further computation in the subroutine. At a later stage of execution of the program after utilizing B, C, D, E registers, there may be a need to restore the original contents to the respective registers. E.g. at the end of the subroutine, the data is restored to the proper register.

The restoration of the contents is a READ operation from the stack and is known as POP operation. A POP register instruction copies the stored data from the stack back into the indicated register pair. Just before the execution of POP instruction, let us say the situation is as shown below:

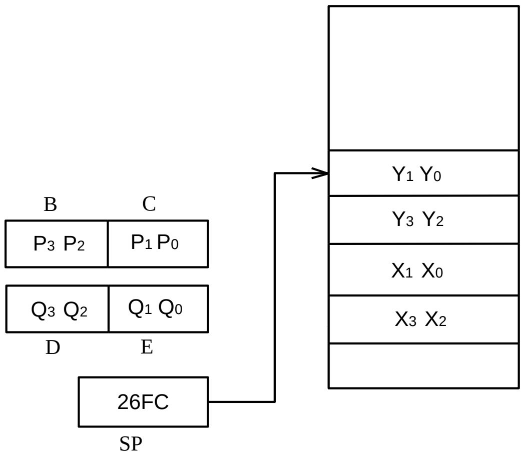


Fig.4.7

Note that registers (B), (C), (D) and (E) have some different contents because these registers are used in the subroutine.

To restore the contents of (B,C) register pair, POP B instruction is used. Whenever this instruction is executed, the contents from the top of the stack are read and written into the (B,C) register pair. To restore the contents (D,E) register pair POP D instruction is used. The question is in which sequence these instructions are to be executed so that the contents are restored properly. The obvious sequence in POP D first & then POP B i.e., the data must be popped off in the reverse order from which it was pushed. This type of stack is called Last-in-First-out (LIFO) memory.

Just after the execution of POP D & POP B instructions, the situation is as shown in figure:

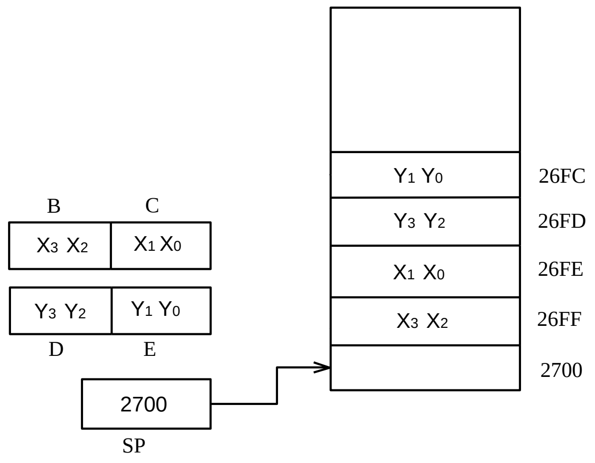


Fig.4.8

When POP D instruction is executed, the data from the top of the stack is copied to register (E), data pointer is incremented by 1, then the next byte of the saved data is copied from the stack to the register (D), and SP is further incremented by 1.

This is similar to earlier status (before PUSH operation) but now some data has been stored in the stack area but these are irrelevant anyway. They will be destroyed during the next PUSH operation on the stack. From the above discussion, following points emerge:

1. The stack pointer always points to the top of the stack up to which it is full with relevant data.
2. Storing or saving the data from the registers on stack is known as PUSH operation.

3. The restoring or reading data from the stack onto certain internal registers is known as POP operation.
4. The stack operates on Last-in-first-out (LIFO) basis.
5. The stack pointer can be initialized to the bottom of the stack but bottom of the stack cannot be utilized to store any useful data.
6. It is for the user to see that the program area does not overlap with stack area.

Lecture-15

W-Z:

(W) and (Z) are two 8-bit temporary registers not accessible to the user. They are exclusively used for the internal operation by the microprocessor. These registers are used either to store 8-bit of information in each (W) and (Z) registers or a 16-bit data in (W,Z) register pair with lower order 8-bits in (Z) and higher-order 8-bits in (W) register.

When a 3-byte instruction containing 2-byte address is to be executed by the μp, the first byte is the (op-code byte) which is fetched and then decoded by the decoder. Then two memories read machine cycles are executed one by one to read the two-byte address, one in each machine cycle and placed in (W,Z) register pair. During instruction execution, in next machine cycle, the address in (W,Z) register pair is transferred to the address latch to address memory or I/O for data transfer.

Increment-Decrement Address Latch:

It is another 16-bit internal register latch available in the register section for internal operations and is not accessible to the user. The address latch serves two functions. First, it selects an address to be sent out from the program counter, from the stack pointer, or from one of the 16-bit register pairs. Second, it latches this address onto the address lines for the required time. The 16-bit addresses from 8085A allow the microprocessor up to 2^{16} memory locations through A_{15}-A_8 and AD_7-AD_0 lines. An increment/decrement register allows the contents of any of the 16-bit registers to be incremented or decremented.

<u>Instruction Register & Instruction Decoder:</u>

The first word of an instruction is the operation code, i.e., binary code for that instruction. Therefore, in the first machine cycle of any instruction μp fetches the instruction from the memory. The op-code representing the instruction to be executed is fetched from the (program) memory location pointed to by (PC) and loaded into the instruction register (IR). The IR passes this op-code to the instruction decoder which interprets this op-code appropriately in order to decide what operation needs to be done for executing this instruction. The instruction decoder tells the control unit the type of instruction to be executed; the number of machine cycles necessary to execute the instruction etc. In response, the control unit generates all the necessary control signals which go into the different internal block of the microprocessor. These different control signals are generated by what is known as Micro-programming technique. Micro-programming means the microprocessor instruction decoding operated like a small version of a μp itself. As the μp goes through the fetch and execute cycles, the microprogramming logic goes through a series of fetch and executes cycles.

E.g. if the instruction is ADI 04H, then the first binary code read by the μp is C6H into the (IR). After decoding this, the decoder will recognize that another memory read cycle is required to read 04H to be added to the number in the accumulator. The decoder will direct the control circuit to send out another memory read pulse and transfers the data coming on the data bus into the temporary register (Temp), so that it can be added to the accumulator. When the addition is completed the control circuit directs the result back to the accumulator. The program counter is then incremented to point the next memory address and send out another memory read pulse to read the μp code of next instruction from memory.

Interrupt Control Section:

Sometimes it is necessary to interrupt the execution of the main program to answer a request from an I/O device. For instance, an I/O device may send an interrupt signal to interrupt control unit to indicate that data is ready for input. The μp temporarily stops what it is doing, inputs the data and then returns to what it was doing. To enable the processor to service the device requesting service through interrupt, processor accepts and issues control signals through interrupt control section.

Serial I/O Control:

Sometimes, I/O devices work with serial data rather than parallel. In this case, the serial data stream from an input device must be converted to 8-bit parallel data before the computer can use it. Likewise the 8-bit data out of a processor must be converted to serial form before a serial output device can use it.

The SID (Serial Input Data) input is where serial data enters the 8085A. The SOD (Serial Output Data) output is where the serial data leaves the 8085A. Two instructions known as SIM & RIM allow the user to perform the serial parallel conversion needed for serial I/O device.

<u>Timing and Control section</u>:

The timing and control section supervise the complete operation of the μp. The on-chip clock oscillator which produces the internal clock is a part of this section. The timing and control section also has a state generator circuit to generate 10 different states namely T_1, T_2, T_3, T_4, T_5, T_6, T_{RESET}, T_{HALT}, T_{WAIT} and T_{HOLD}. State generator is a multi-mode counter. The next state of the state generator from the present state is decided by many of the control signals input like READY, HOLD, Interrupt control signals - TRAP, RST7.5, RST6.5, RST5.5 and INTR. In each state this section generates many control signals for executing the instruction fetched.

The operation of the μp is cyclic in natural. During the normal operation from the word GO, μp sequentially fetches and executes one instruction after another until a HALT instruction is executed. The fetching and execution of a single instruction constitutes an instruction cycle. The instruction cycle consists of one or more read or write operation to memory or an I/O device. Each memory and I/O reference requires a mechanic cycle. In other words every time a byte of data is move from CPU to I/O or memory or from memory or I/O to CPU, a machine cycle is required.

There are seven different kinds of machine cycles in the 8085 A:

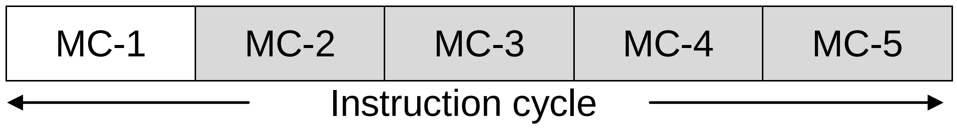
1. Opcode Fetch Machine Cycle (OFMC)
2. Memory Read Machine Cycle (MRMC)

3. Memory Write Machine Cycle (MWRMC)
4. I/O Read Machine Cycle (IORDMC)
5. I/O Write Machine Cycle (IOWRMC)
6. Interrupt Acknowledge Machine Cycle (INTAMC)
7. Bus Idle Machine Cycle (BIMC)

Three status signals $IO/\overline{M}$, S_1 and S_0 generated at the beginning of each machine cycle and $\overline{RD}$, $\overline{WR}$ and $\overline{INTA}$ generated during T_2 state of the machine cycle identify each type of the machine cycle. The status signals remain valid for the entire duration of the cycle. The instruction fetch portion of an instruction cycle requires a machine cycle for each byte of the instruction to be fetched. Since an instruction consists of 1 to 3 bytes (1, 2 or 3), the instruction fetch is one to three machine cycles in duration.

The first machine cycle of an instruction cycle is always an OPCODE FETCH machine cycle which is always single byte long and the 8-bits obtained during an OPCODE FETCH are always interpreted as an OPCODE of an instruction. Note that to fetch an instruction, i.e., to transfer an entire instruction from memory to the μp necessitates an OPCODE FETCH machine cycle. However, one or two memory read machine cycles are also needed to complete the fetch for 2^{nd} and 3^{rd} bytes of the instruction respectively.

The number of machine cycles required to execute the instruction depends on the particular instruction. Some of the instructions require no addition machine cycles after the instruction fetch is complete, other requires additional machine cycles to write or read data to or from memory or I/O devices. The total number of machine cycles required varies from one to five. Around 50% of the instructions require only one machine cycle for fetching and executing the instruction. No instruction requires more than five machine cycles. Machine cycles like the memory read or memory write may occur more than once in a single instruction cycle.



The shaded area may be required for executing the instruction. The timing and control unit of μp automatically generates the proper machine cycles required for an instruction cycle from information provided by the op-code.

Each machine cycle contains a number of 320ns clock periods when cryptal used is 6.25 MHz. One clock period, i.e. the period between two negative going transitions of that clock is called T state. The various T-states are T_1, T_2, T_3, T_4, T_5 and T_6. Most of the machine cycles have three T-states each (T_1, T_2, T_3). Only OPCODE FETCH machine cycle has either 4 or 6 states depending on the instruction. The first 3rd states of the machine cycle are identical to a MRMC, the additional T states in OFMC are the T-states required by the 8085A to decode the op-code and decide what actions are needed in succeeding machine cycles.

The combined MCS along with T-states are shown in fig.

MC-1						MC-2			MC-3			MC-4			MC-5		
T_1	T_2	T_3	T_4	T_5	T_6	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3

MC-i (i=2, 3, 4, 5)

Thus one complete transition from state T_1 through the state diagram and back to T_1 constitutes a complete machine cycle. The partial state transition diagram is shown in fig.4.9 assuming READY=1 i.e., no wait state.

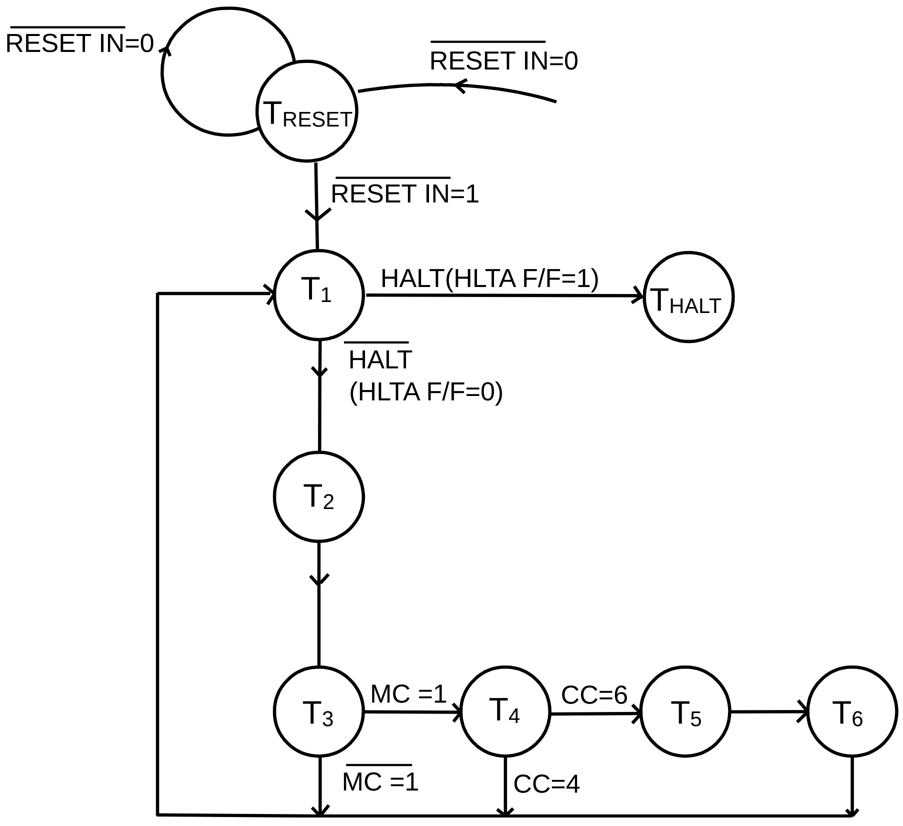


Fig.4.9 Partial State Transition Diagram indicating T_1-T_6 States

The shaded portion shows that these states may be needed in some instructions. Instruction cycles for various 8085A instructions require from 4 to 18 states. The total number of states actually required to execute an instruction will depend on the READY & HOLD signal inputs.

For example, consider the 3 byte instruction

STA ADDR.

STA stands for store accumulator direct. The meaning of the instruction is transfer the content of the accumulator to an external memory location whose address is specified in the instruction i.e., ADDR. Since this location can be anywhere in the 64k memory space that the 8085A can directly address, 16-bits are required for the address. Thus the instruction contains 3 bytes- a 1 byte op-code and 2-byte address. The instruction is stored in the memory as follows:

OP CODE	Byte -1
LOWER ADDR	Byte - 2
HIGHER ADDR	Byte - 3

Three machine cycles (MC) are required to fetch this instruction. In MC-1, i.e., Op-code fetch machine cycle, the op-code is transferred from memory to the instruction register during T_1-T_3 states and then during T_4 state it is interpreted. At this point, the CPU knows that it must do more machine cycles - two MRMCs to fetch the complete instruction. In MC-2 the lower address is transferred from the memory to the temporary register (Z). In MC-3 the third byte, i.e. the higher byte address is transferred from the memory to the temporary register (W). When the entire instruction is in the μp, it is executed. Execution means a data transfer from the μp to memory. The content of the accumulator is transferred to the memory location, whose address was previously transferred to the μp by the proceeding two memory read machine cycles. The address of the memory location to be written is generated as follows:

The high order address byte in temp register (W) is transferred to the address latch and the low order address byte in temp register (Z) is transferred to address/data latch. The content of the (A) is then placed on the data bus. This data transfer is affected by a MWRMC.

Thus 3-byte STA instruction has four machine cycles in its instruction cycles.

Mnemonic	Instruction byte	Machine Cycle
STA	OP code	OFMC
	Lower address	MRMC
	Higher address	MRMC
		MWRMC

The actions taken by the processor in different machine cycles are shown in fig.4.10.

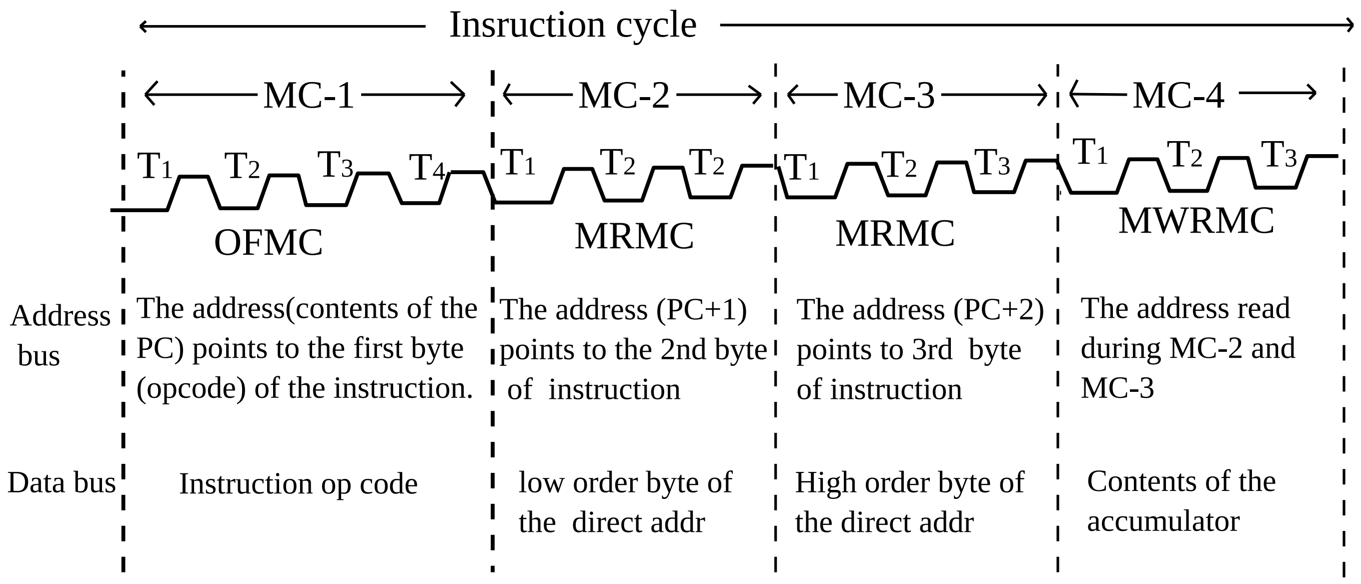

Insruction cycle
MC-1
MC-2
MC-3
MC-4
T1 T2 T3 T4
T1 T2 T2
T1 T2 T3
T1 T2 T3
OFMC
MRMC
MRMC
MWRMC
Address bus
The address(contents of the PC) points to the first byte (opcode) of the instruction.
The address (PC+1) points to the 2nd byte of instruction
The address (PC+2) points to 3rd byte of instruction
The address read during MC-2 and MC-3
Data bus
Instruction op code
low order byte of the direct addr
High order byte of the direct addr
Contents of the accumulator

Fig.4.10 Instruction Cycle and Machine Cycles for STA Addr Instruction

Thus STA ADDR instruction has a total of 13 states. If the 8085A is operating at 325.5ns time, the STA instruction cycle is executed in 4.23 μsec. This time period is the instruction execution time, although it actually includes both the instruction fetch and the execution time.

Lecture-16

OPCODE FETCH Machine Cycle:

Figure shows the 8085A instruction fetch timing diagram. The instruction fetch cycle requires either four or six clock periods (T-states). The other machine cycles that follow OFMC will need three clock cycles.

The purpose of an OFMC is to read the contents of a memory location containing the opcode addressed by the program counter and to place it in the instruction register (IR).

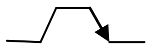
In the beginning of state T_1, the 8085A puts a low on the $IO/\overline{M}$ line of the system bus indicating a memory operation. The 8085A sets S_1=1 and S_0=1 on the system bus, indicating the memory fetch operation. This status information remains available for the duration of the machine cycle. During T_1 state, the 16-bit address A_{15}-A_0 of the memory location containing the opcode is obtained from the program counter (PC) and placed on the address and address/data latches. The higher order 8-bits of the address appear on the address bus A_8-A_{15} remains constants until the end of the state T_3. During T_4 state the data on the address bus is unspecified. The low order 8-bits of the address are placed on the address/data bus, AD_7-AD_0 at the beginning of T_1. This data however remains valid only until the beginning of state T_2 at which time the address/data bus is floated (tri-stated) because this is time multiplexed bus and used as the data bus during T_2 and T_3 states. Therefore address latch enable (ALE) signal issued by the μp during T_1 is used to latch this lower order address in some external latch 74LS373 on its falling edge. The 16-bit address selects a particular memory location.

During state T_2, at the beginning, the $\overline{RD}$ signal goes low indicating read operation and the opcode to be fetched is placed on the data bus, AD_7-AD_0 by the addressed memory location. The contents of (PC) is incremented be 1 during this state as during T_1 state the (PC) has sent the address to address bus. The accessed memory should be fast enough to output its data before $\overline{RD}$ goes high. Slower memories can gain more time by pulling the READY signal of 8085A LOW. This will introduce an integral number of T_{WAIT} states between T_2 and T_3 as long as READY is low. On the rising edge of the $\overline{RD}$ control signal in T_3 state, the opcode obtained from the memory is transferred to the microprocessor instruction register.

During state T_4, the 8085A decodes the instruction and determines whether to enter state T_5 or to enter T_1 state of the next machine cycle. From the operation code, the μp determines what other machine cycles, if any, must be executed to complete the instruction cycle. State T_5 and T_6 when entered, are used for internal μp operations necessitated by the instruction.

The micro RTL flow for 4-states OFMC is shown below.

<u>OFMC:</u> Status signals IO/$\overline{M}$=0, S_1=1, S_0=1

T_1: A_{15}-A_8 $\longleftarrow$ (PCH), AD_7-AD_0 $\longleftarrow$ (PCL), ALE = 

T_2: $\overline{RD}$ = 0, (PC) $\longleftarrow$ (PC) +1, AD_7-AD_0 $\longleftarrow$ M(AB)

T_3: $\overline{RD}$ = 1, $\uparrow$, (IR) $\longleftarrow$ BDB

T_4: μp decodes the opcode and decides whether T_5 and T_6 states are required or next machine cycle executed is T_1

During T_2 state, after the $\overline{RD}$ signal is made LOW, the external decoding circuit decodes the address put on the address bus duirng

T_1 state. One of the memory location is selected and it puts 8-bit information on the data bus during T_2 and T_3 states. Processor has no control on it. Processor has already issued the signals and now it is the job of the external decoding circuit to make use of the signals $IO/\overline{M}$ and $\overline{RD}$ and address lines to allow the external memory to put the data on the data bus. Therefore, this action is shown by shaded area. Whatever information is avaiable on BDB at LOW to HIGH transition of $\overline{RD}$, that will be read and processed. The timing waveform during 4-state OFMC is shown in fig.4.11.

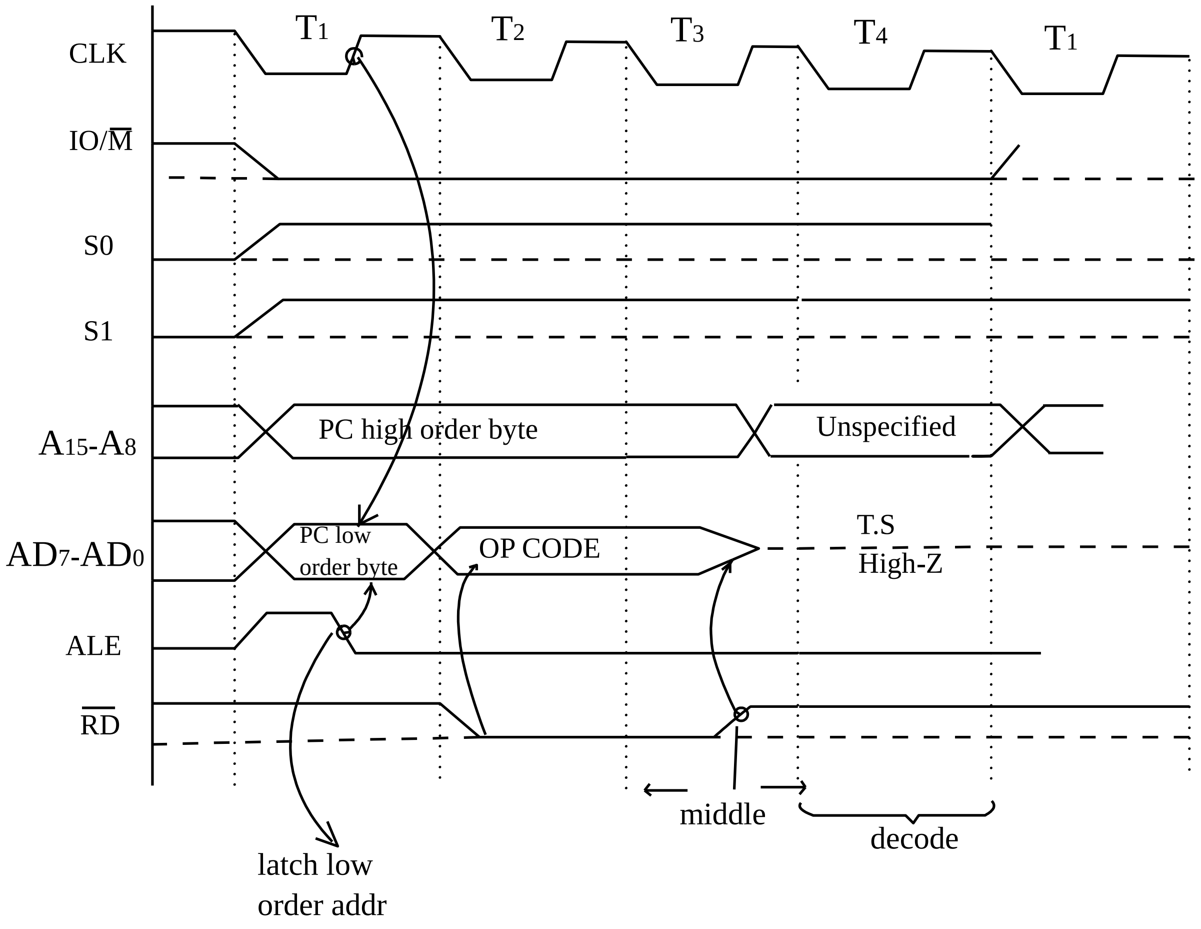


Fig.4.11 Timing Diagram During 4-state OpCode Fetch Machine Cycle

During T_4–T_6 states, AD_7-AD_0 lines are tri-stated and A_{15}-A_8 lines are unspecified.

Fig.4.12 shows the timing diagram for a 6-state OFMC:

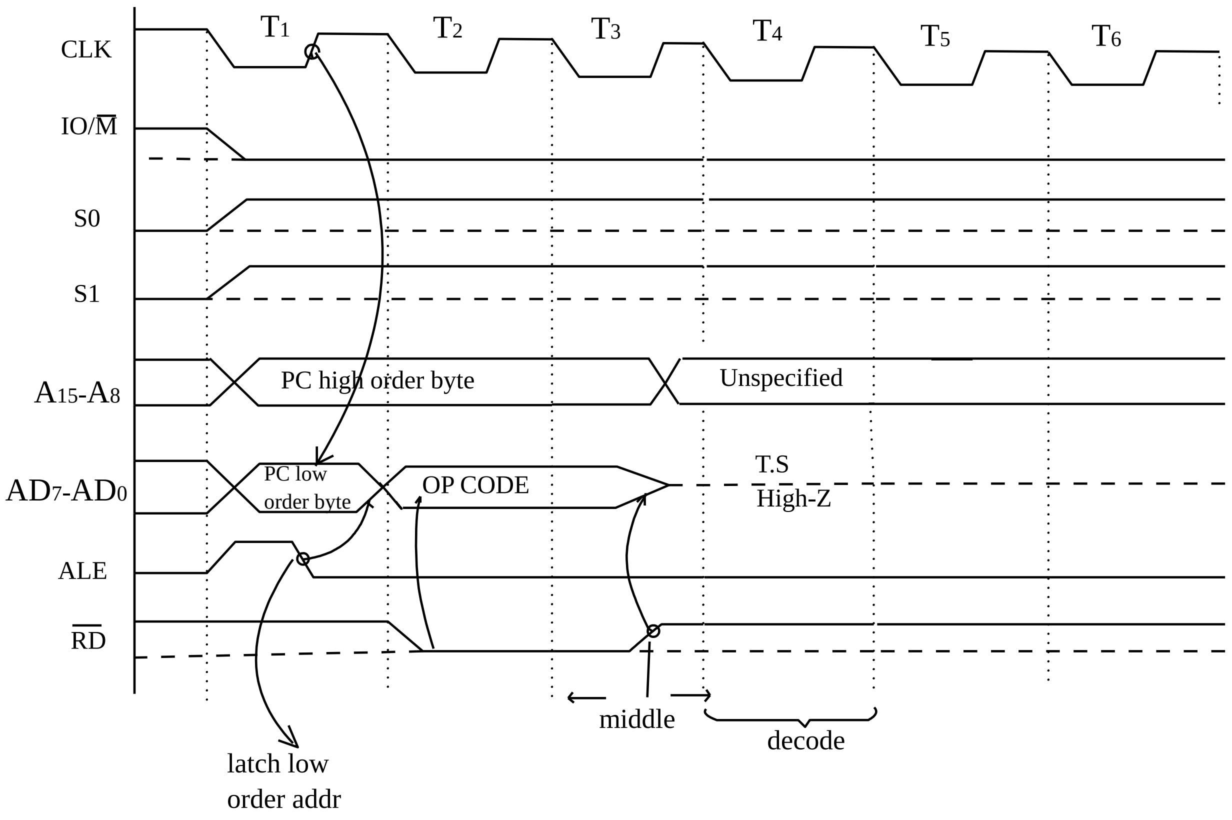


Fig.4.12 Timing Diagram During 6-state OpCode Fetch Machine Cycle

Note: Whenever the address information is sent from the program counter to the external world during T_1 state, then the (PC) is incremented by 1 during the subsequent T_2 state so that PC points to the next subsequent byte. However, if the address information from (PC) has not been sent out during the T_1 state to the external world, then (PC) will not be incremented during T_2 state.

Memory READ Machine Cycle:

It requires 3 states T_1 to T_3. The purpose of the memory READ operation is to read the contents of a memory location addressed by a register pair and place the data in one of internal registers of the

μP. The source of address issued during T_1 is not always the program counter but may be any one of the several other register pairs in the μP depending on the particular instruction of which the machine cycle is a part.

The 8085A uses machine cycle MC-1 to fetch and decode the instruction. It then performs the memory read operation in MC-2. E.g. in LXI H, Addr.

The IO/$\overline{M}$ signal is made LOW to indicate the external world that a memory reference is required. Then μP made S_0=0 and S_1=1 indicating that memory READ operation is to be performed. During T_1, the μP places the contents of higher byte of the memory address register, such as that contents of the (PCH) or (H) register on A_{15}-A_8 and the contents of the lower byte of the memory address register such as contents of the (PCL) or (L) register on AD_7-AD_0. The μP sets ALE signal HIGH indicating the beginning of MC-2. As soon as ALE goes to LOW in the middle of T_1, the lower byte of the address is latched in an external latch. The same bus is now going to be used as data bus.

During T_2 state, the $\overline{RD}$ signal goes LOW indicating a READ operation. If the address sent out during T_1 state is from (PC), then (PC) is incremented by 1 otherwise not. The external logic gets the data from the memory location addressed by the memory address register such as (H,L) pair and places the data on to bi-directional data bus AD_7-AD_0.

During T_3 state, $\overline{RD}$ signal goes HIGH. This LOW to HIGH transition of signal transfers the data from the data bus to internal register such as the accumulator.

MRMC: Status signals $IO/\overline{M}=0$, $S_1=1$, $S_0=0$

T_1: A_{15}-A_8 $\leftarrow$ (PCH), AD_7-AD_0 $\leftarrow$ (PCL), ALE =

T_2: $\overline{RD}$ = 0, (PC) $\leftarrow$ (PC) +1, AD_7-AD_0 $\leftarrow$ M(AB)

T_3: $\overline{RD}$ = 1, $\uparrow$, (Internal Reg.) $\leftarrow$ AD_7-AD_0 or BDB

Or

T_1: A_{15}-A_8 $\leftarrow$ (H), AD_7-AD_0 $\leftarrow$ (L), ALE =

T_2: $\overline{RD}$ = 0, AD_7-AD_0 $\leftarrow$ M(AB)

T_3: $\overline{RD}$ = 1, $\uparrow$, (Internal Reg.) $\leftarrow$ AD_7-AD_0 or BDB

The timing diagram during memory ready machine cycle is shown in fig.4.13.

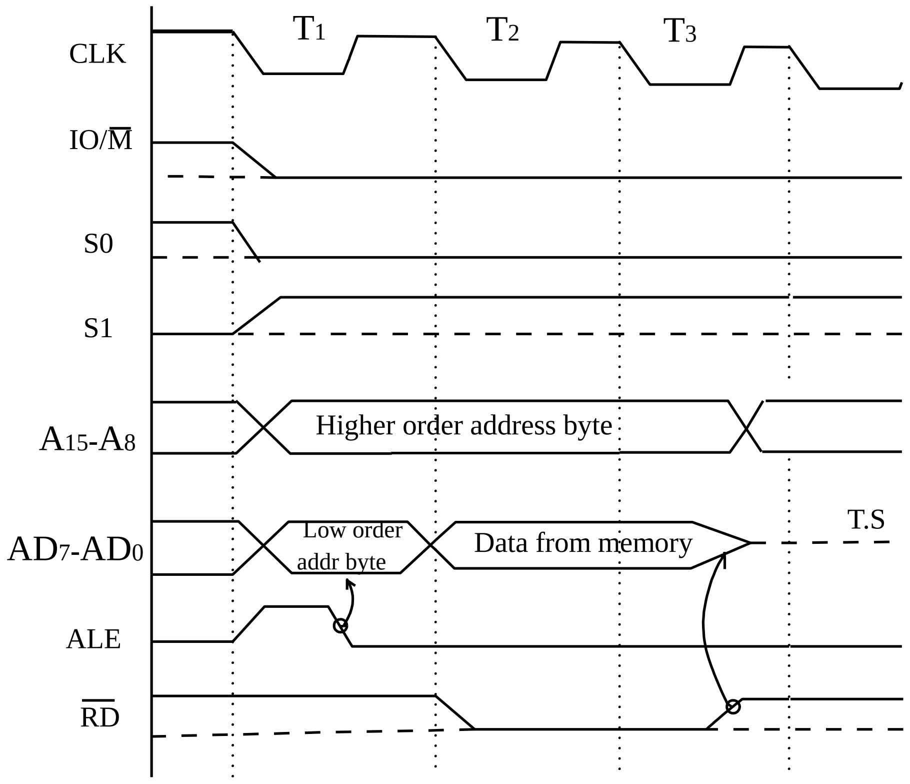


Fig.4.13 Timing Diagram During Memory Read Machine Cycle

Lecture-17

Memory WRITE Machine Cycle:

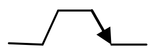
It also requires only T_1 to T_3 states. The purpose of memory write machine cycle is to store the contents of any of the 8085A register such as the accumulator into a memory location addressed by a register pair such as (H,L).

The 8085A μP made $IO/\overline{M} = 0$ in the beginning of T_1 state to indicate memory reference operation. Then it puts $S_0 = 1$ and $S_1 = 0$ indicating a memory write operation.

During T_1 state 8085A places the memory address register (MAR) higher byte such as the contents of the (H) register on lines A_{15}-A_8 and also places the MAR lower byte such as the contents of the (L) register on lines AD_7-AD_0. The μp sets ALE signal HIGH indicating the beginning of MWRMC. As soon as ALE goes to low, the lower byte of the address is latched in an external latch. During T_2 state, $\overline{WR}$ goes LOW indicating memory write operation. It also places the contents of the internal register, say accumulator, on data lines AD_7-AD_0.

During T_3 state, $\overline{WR}$ goes HIGH. This LOW to HIGH transition is used to transfer the data from the data lines to the memory location address by MAR such as (H,L) register pair.

MWRMC: Status signals $IO/\overline{M}=0$, $S_1=0$, $S_0=1$

T_1: A_{15}-$A_8 \longleftarrow$ (H), AD_7-$AD_0 \longleftarrow$ (L), ALE = 

T_2: $\overline{WR} = 0$, AD_7-$AD_0 \longleftarrow$ (μp Internal Reg.)

T_3: $\overline{WR} = 1, \uparrow$, M(AB)$\longleftarrow AD_7$-AD_0 or BDB

Similar to MRMC, the processor simply puts the data on the data bus and makes required signals LOW or HIGH. It is the job of the external decoding circuit to make use of these signals to enable the external memory to accept the data from the data bus. Processor has no control over it. Therefore, this action during T_3 state is shown shaded. The timing diagram during MWRMC is shown in fig.4.14:

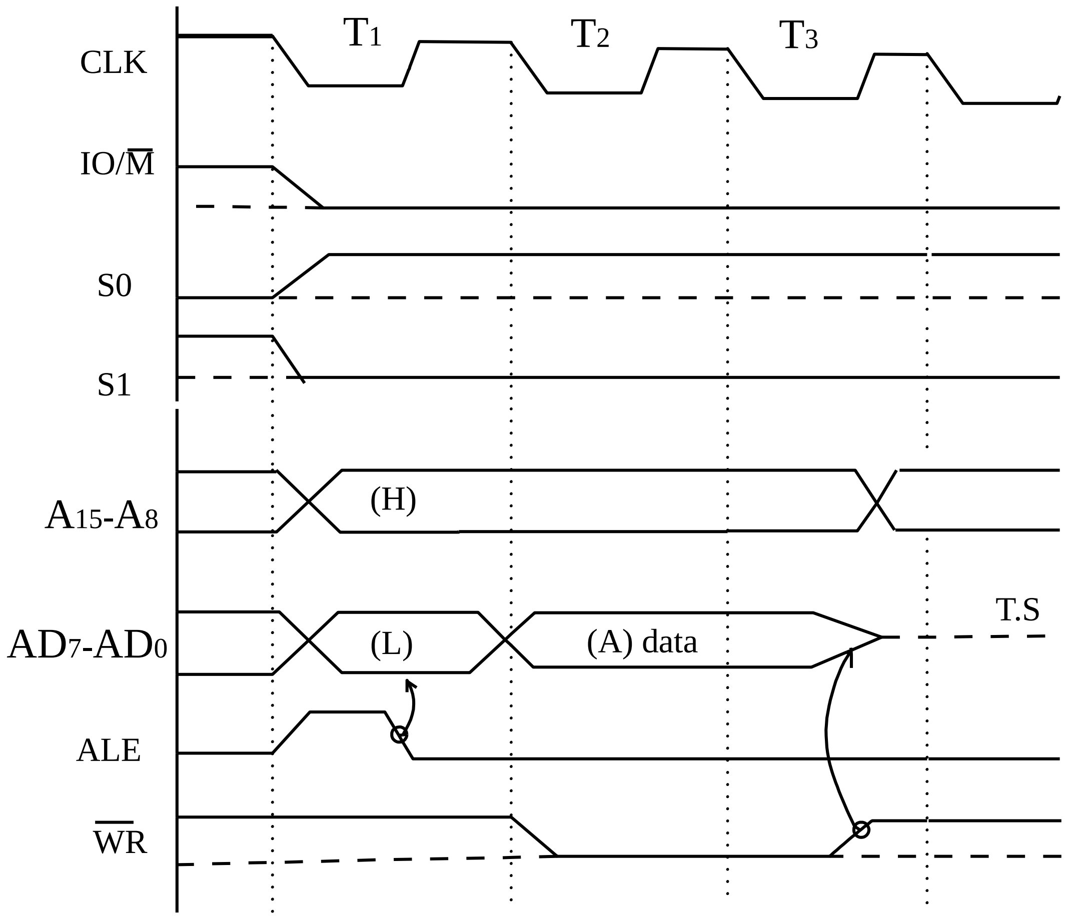


Fig.4.14 Timing Diagram During Memory Write Machine Cycle

<u>I/O READ and I/O WRITE M/C cycle</u>:

The IORDMC and IOWRMC are identical to MRMC & MWRMC respectively except that appropriate status signals are issued at the beginning of T_1 state. IO/$\overline{M}$ signal goes HIGH at the beginning to indicate I/O device reference is needed in case of I/O mapped input/output device. In these machine cycles higher & lower address bytes are identical and equal to the 8-bit address of the I/O port while in case of MRMC or MWRMC, the address bus output is the true 16-bits address. These machine cycles will be discussed in detail alongwith I/O techniques.

<u>HALT State:</u> (T_{HALT}):

Whenever the HLT instruction is executed μp enters in to the HALT state. The opcode for HLT instruction is 76H. Assume that an opcode fetch machine cycle is initiated, and the opcode transferred to the instruction register during T_3 state is 76H i.e, the opcode of HLT instruction. During state T_4, the control unit decodes the instruction opcode and sets an internal HALT flip-flop of the processor. Upon exiting state T_4, the μp enters state T_1 of the next machine cycle. As indicated in the figure, the HALT flip-flop is checked in T_1 state of the next machine cycle. If it is found set, instead of entering T_2 state, the μp enters in to the HALT state otherwise in T_2 state. Thus five states are required to reach the HALT state. In the HALT state, the address and address/data buses along with $\overline{RD}$, $\overline{WR}$ and IO/$\overline{M}$ are placed in their high independence states (floated).

There are only three ways to exit from a HALT state as shown in fig.4.15.

1. A LOW an $\overline{\text{RESET IN}}$ input of the 8085A resets the entire system and loads the program counter with all 0's. When $\overline{\text{RESET IN}}$ signal is active, μp comes out of HALT state and enters into

RESET state and remains their as long as $\overline{\text{RESET IN}}$ is active. After reset, 8085A immediately starts program execution from 0000H.

2. The second way to get out of the HALT state is to make the HOLD signal input high. The processor then enters the HOLD state, but when the HOLD input goes LOW again, the CPU returns to the HALT state.
3. The third method of coming out of a HALT state is when and interrupt signal is active. This method works only if interrupts were enabled with an enable interrupt (EI) instruction in the program before HALT instruction is executed. Whenever interrupt comes μp leaves the HALT state and start executing the interrupt service subroutine (ISR).

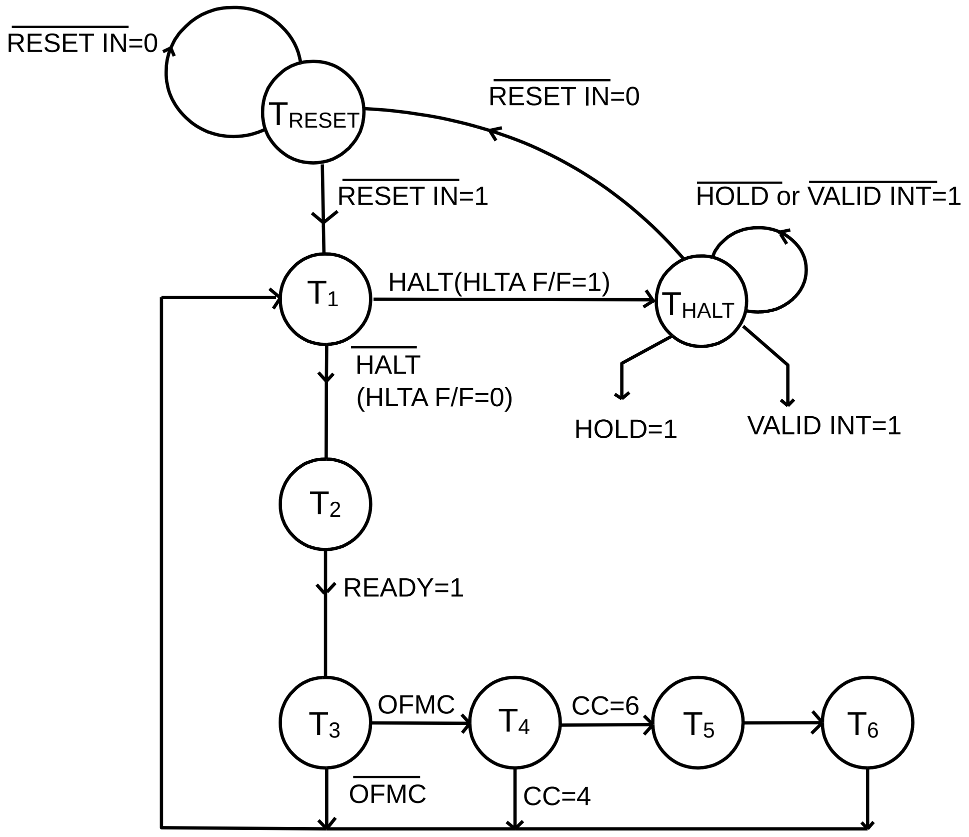


Fig.4.15 Partial State Transition Diagram indicating HALT State

WAIT State T_{WAIT}:

According to timing specification for the 80854A, during a read operation (OFMC/ MRMC/ IORMC), the device providing data to the μp must have valid data on the dater bus within [(5/2) T-225] ns after the μp provides a valid address at its address pins. For T=320ns, the memory or input device must have an access time of 575ns or less.

Sometimes microprocessors are used with memories or I/O devices which have longer access time. In case of memories, the lower the cost, generally the longer the access time. To accommodate long access time, the longer the access time. To accommodate long access time, the 8085A has a state called the WAIT state, T_{WAIT} as shown in fig.4.16

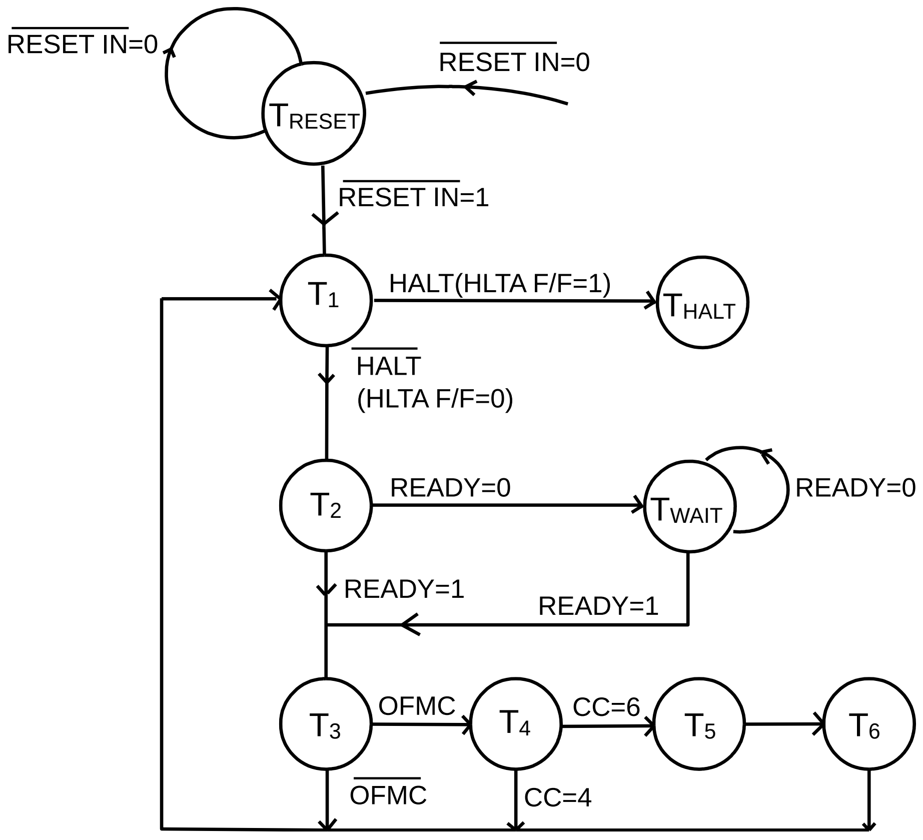


Fig.4.16 Partial State Transition Diagram indicating WAIT State

When the μp places the address of the memory or I/O device on address bus in T_1 state, external control logic monitoring this address can request that the microprocessor waits for a period of time equal to an integral number of clock periods. The external control logic does this by making the READY input signal to the μp logic '0' during T_2 state. After making $\overline{RD}$ signal LOW in T_2 state, microprocessor monitors READY signal input. If this input is found LOW, then microprocessor enters in T_{WAIT} state instead of T_3. When the READY signal becomes logic '1', the μp comes out of T_{WAIT} state and enters into T_3 state and machine cycle continues. Wait states continue to be inserted as long as READY is LOW.

The effect of entering a wait states is to hold all external signals from the μp in the same state they were on at the end of state T_2 i.e., the content of address bus, data bus, and control bus are all held constant. This stretches the duration of address and $\overline{RD}$ pulse, so devices with access time greaten than 575ns can be read. If N wait states are introduced into the machine cycle, the required access time is [(5/2 + N) T-225] ns.

Fig.4.17a shows a circuit to insert single WAIT state in OFMC.

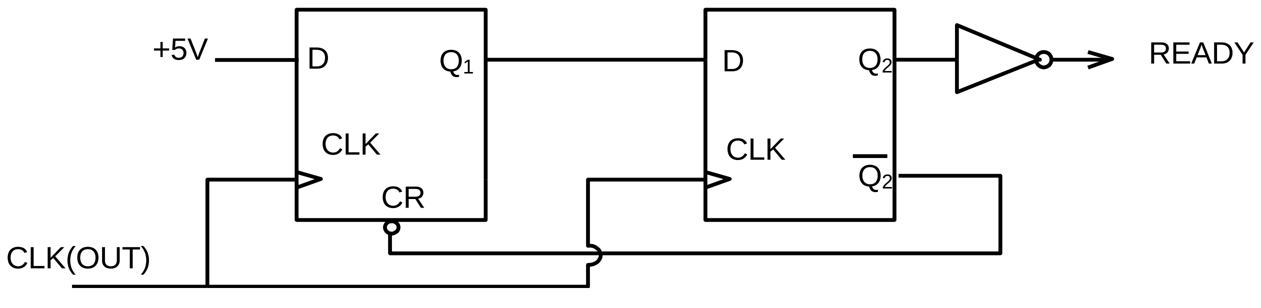


Fig.4.17(a) Logic Circuit to Control READY Signal Input

The waveforms at different points of control circuit alongwith address bus, data bus and control signals are shown in fig.4.17b.

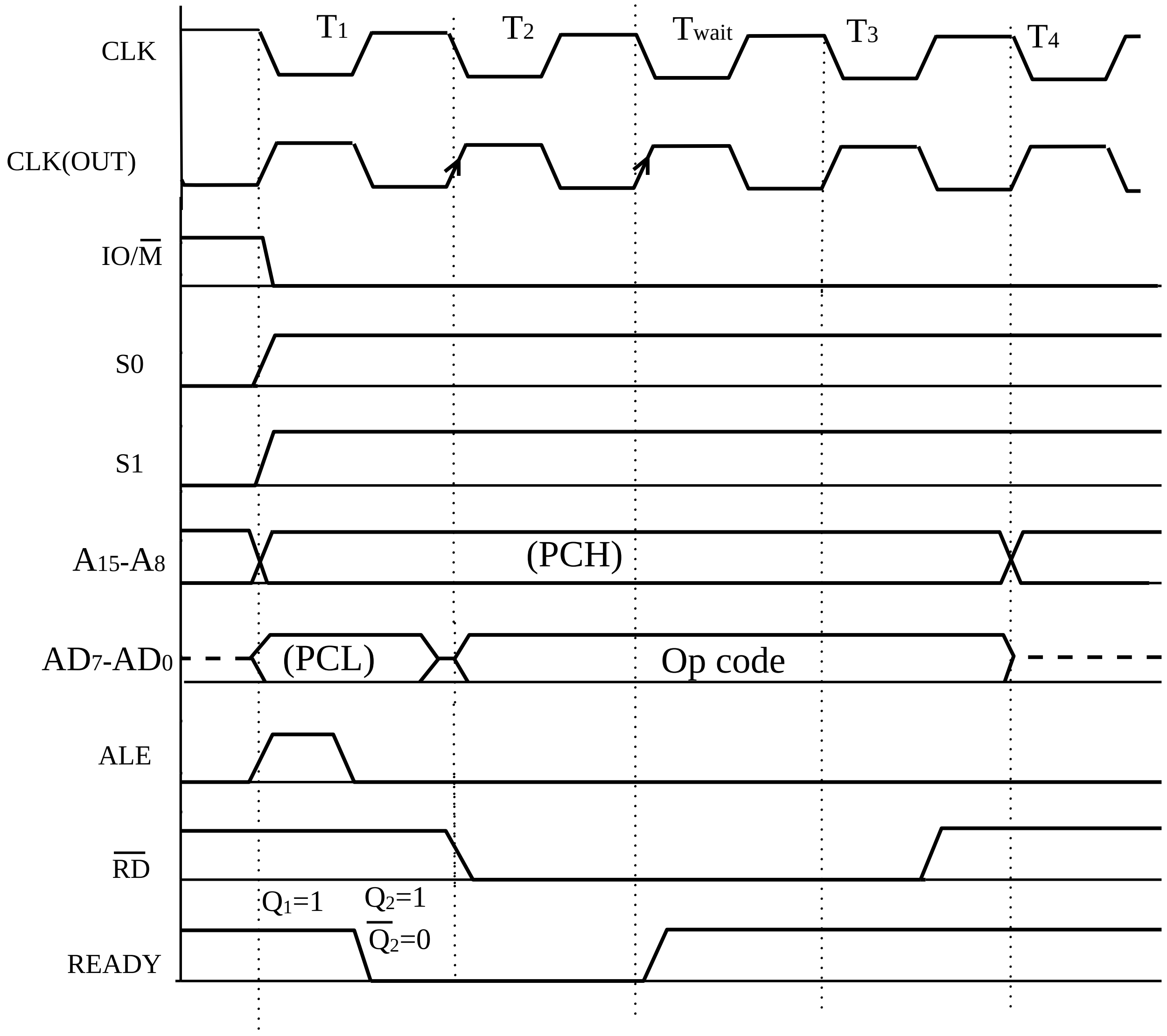


Fig.4.17(b) Waveforms at Different Points to Insert Single WAIT State

The CLK(OUT) signal is 180° out of phase with CLK signal. The rising edge of CLK signal sets Q_1 and therefore, D input of 2nd flip-flop. Before the processor checks the READY signal during T_2 state, rising edge of CLK(OUT) signal makes the READY signal LOW. Sampling of the READY line in state T_2 inserts WAIT state The $\overline{Q}_2$

output also clears 1st flip-flop and Q_1 becomes LOW. The next CLK(OUT) signal sets the READY signal. Sampling of the READY line again in WAIT state allows processor to enter in T_3 state. Thus a single WAIT state is inserted in OFMC and allows the μp to synchronize to memories or I/O devices with long access time. This, of course, is associated with increased instruction cycle time and additional logic to control the READY input.

External logic controlling the READY line can be designed so that none, a fixed number or a variable number of WAIT states transitions occur during each cycle. This logic can also be designed so that these wait states occurs only for specific types of machine cycles eg. OFMC.

A monostable can be triggered by 8085A $\overline{RD}$ or $\overline{WR}$ pulse as shown in fig.4.18, to make READY signal LOW each time the slower device is addressed. The monostable can be enabled by the same signal that is sent to select the addressed device. This prevents a WAIT state to be introduced during each read or write operation.

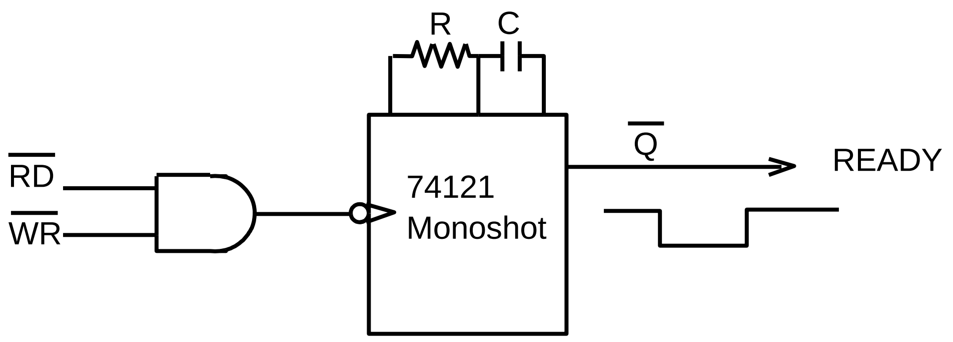


Fig.4.17 Monoshot Used to Make READY Signal LOW for Fixed Duration

Lecture-18

HOLD State T_{HOLD}:

The μp enter in this state, if some external device wants direct memory access (DMA) so that a string of data can be transferred to or from the memory at a fast rate. The device requesting for DMA makes the HOLD signal input high.

There are two possibilities-(i) the μp may be in the HALT state and then the HOLD signal input becomes HIGH. In this situation, the μp first sets the HLDA flip-flop (HOLD acknowledge F/F) & then enters the HOLD state (ii) the μp is executing some machine cycle. While execution μp checks the HOLD signal at unique points during a machine cycle. HOLD request signal to processor is asynchronous in nature. The μp synchronizes this request and at proper time in a machine cycle provides the HLDA signal by setting HLDA F/F to the requesting device. The HOLD state is entered after a machine cycle is completed.

The HOLD signal is checked during T_2 state (after READY input has been checked) and also during T_4 state (provided the concerned machine cycle requires T_5 & T_6 states also). If the HOLD signal is found high, HLDA F/F is set and the processor enters the HOLD state after the current machine cycle is over.

Upon entering the HOLD state, the HLDA output signal from the μp is set HIGH. During this state, the address & the data buses at the $\overline{RD}$, $\overline{WR}$ & $IO/\overline{M}$ control lines are floated (tri-stated). By floating its address, data and control buses, the μp effectively disconnects itself from the system. From this point on, it is up to the requesting device to provide address, data & control signals to memory & I/O port to implement the data transfer i.e., the requesting device then enables its tri-state buffers. When DMA process is over it floats its address data, and control buses and then bring the HOLD signal input LOW.

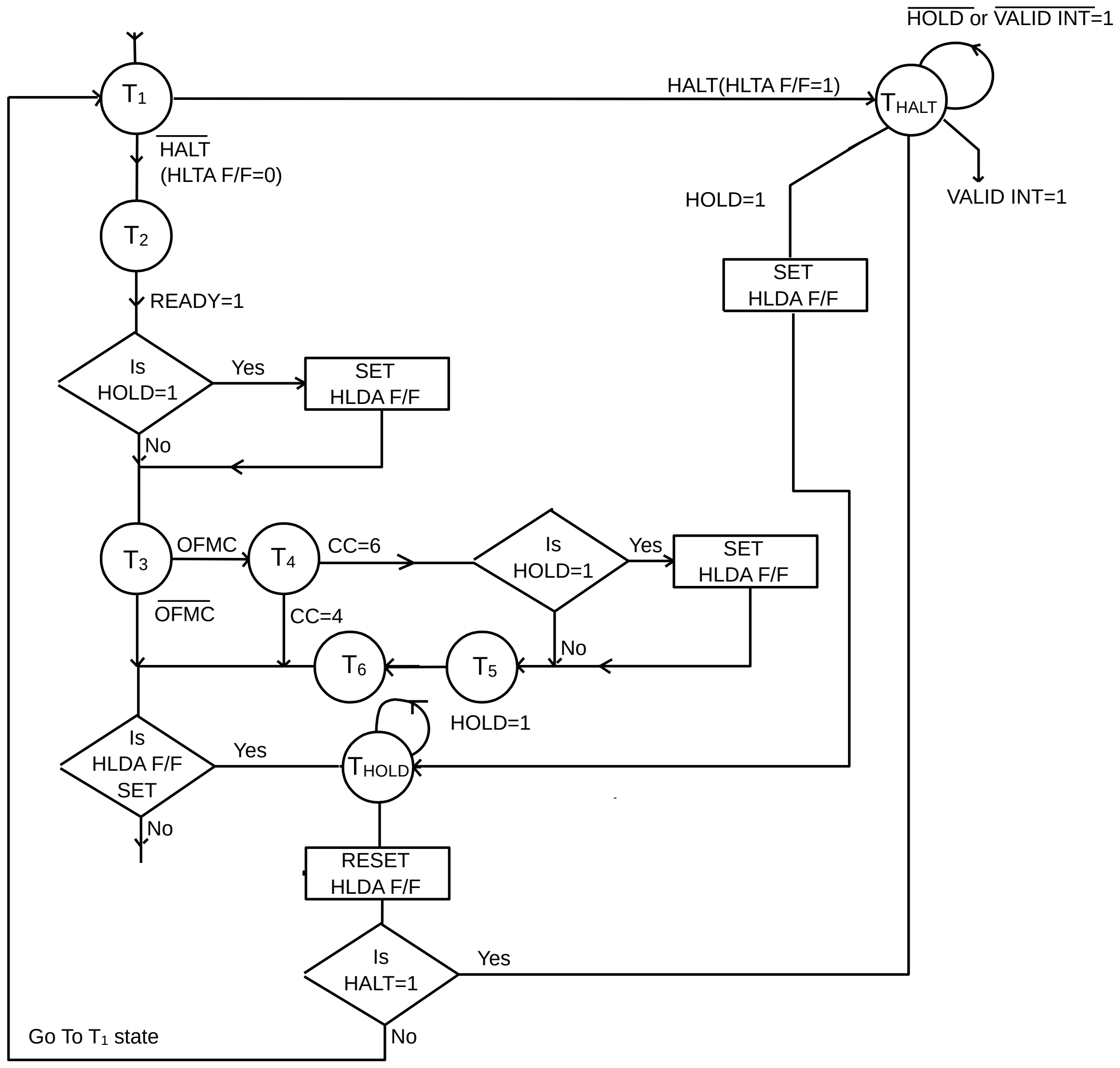


Fig.4.19 Partial State Transition Diagram Indicating HOLD State

The μp exits the HOLD state and then continue its previous operation from the point at which it was suspended by the HOLD request. If resets, the HLDF F/F first. If the HALT F/F is found to be set, it enters the HALT state else enters T_1 state. The partial state transition diagram showing HOLD state is shown in fig.4.19.

STATE TRANSITION DIAGRAM

Figure shows the complete state transition diagram of 8085A. As discussed in previous lecture, the processor will be in one of the 10 different states namely T_1, T_2, T_3, T_4, T_5, T_6, T_{RESET}, T_{HALT}, T_{WAIT} and T_{HOLD}. The next state of the state generator from the present state is decided by many of the control signals input like READY, HOLD, Interrupt control signals - TRAP, RST7.5, RST6.5, RST5.5 and INTR. State transition diagram is a compact way of showing when during an instruction cycle the 8085A will enter a HALT state, insert a WAIT state, respond to HOLD input or respond to an interrupt input.

If $\overline{\text{RESET IN}}$ is asserted, the μp stays in a reset state with the address bus floating. When $\overline{\text{RESET IN}}$ is not asserted or the previous machine cycle is finished, the CPU enters T_1 of a new machine cycle. If the previous instruction executed was a HLT instruction state, the CPU goes directly to a HALT state. The three ways to exit the halt state are by a RESET, a valid interrupt and a HOLD request. Note that the exit from HALT state is only temporary. As soon as HOLD is not asserted, the CPU returns to the halt state.

If a halt state was not entered, then the CPU proceeds to T_2 of the machine cycle. Here it checks the READY input. If the READY line is not asserted (LOW), the CPU inserts wait states until READY goes high.

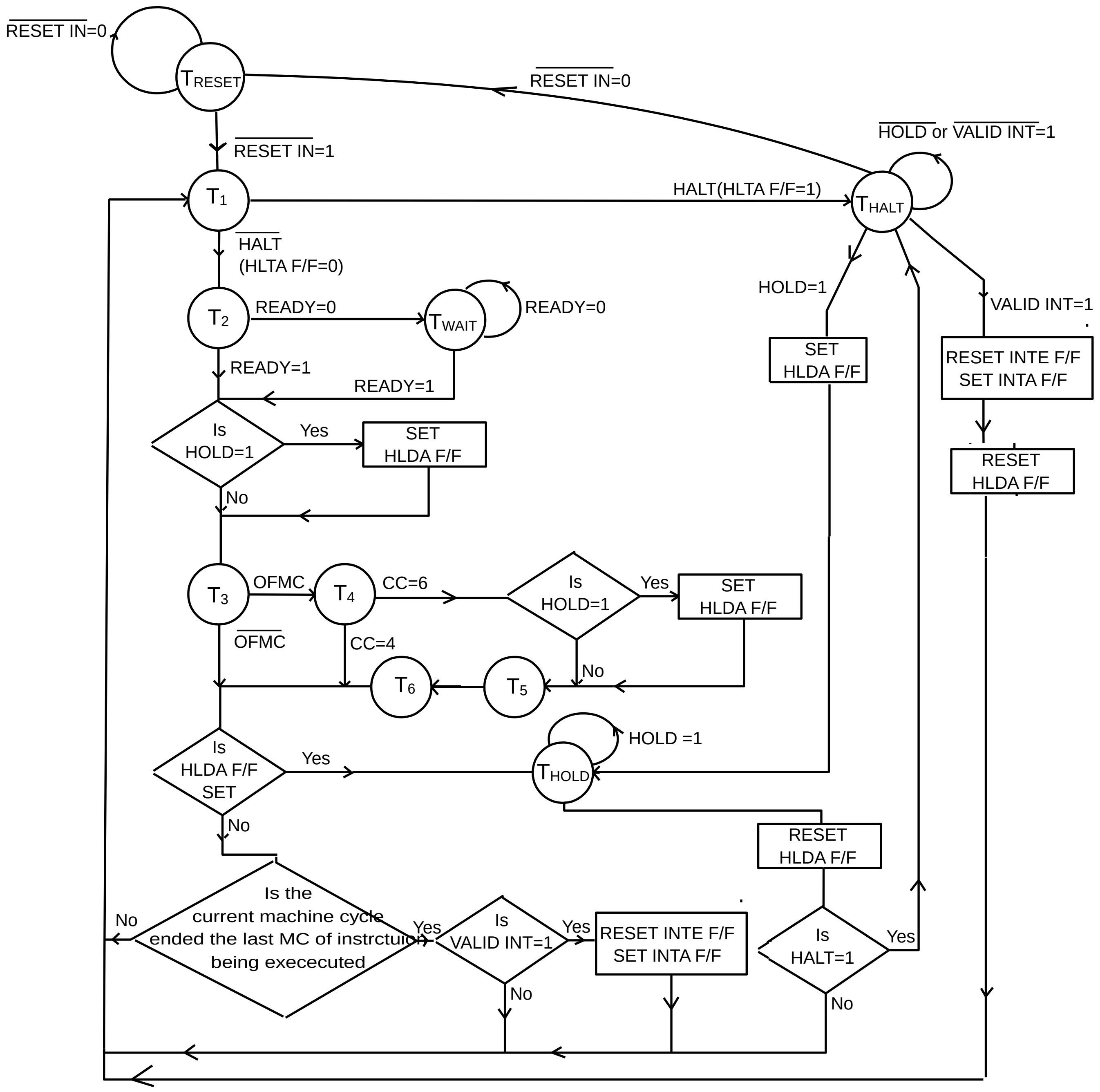


Fig. State transaction diagram of 8085 A μP

The hold input is checked at several points. If a hold request is present on it, the hold-acknowledge flip-flop is set. However the CPU will not enter the HOLD state until the end of the current machine cycle.

Note that the μp does not check whether a valid interrupt request is present until the end of the instruction cycle. This is necessary so that the address of the next instruction can be pushed onto the stack. The processor checks valid interrupt after the completion of instruction cycle because it has to execute the interrupt service subroutine and for that internal registers, PC, SP are required. However, it checks the HOLD signal after every machine cycle and enters into HOLD state at the end of current machine cycle if the signal is active. In HOLD state processor has nothing to do – neither memory read/write nor I/O operation. Therefore, if both HOLD and Interrupt becomes active together, processor first respond to HOLD signal and after DMA operation only it responds to interrupt. The complete state transition diagram is shown in fig.4.20.

To summarize, a halt state is entered during the $T_{1,}$ a wait state is entered after T_2, a hold state is entered after a machine cycle is completed, and an interrupt is responded to after an instruction cycle is completed.