MARUDHAR KESARI JAIN COLLEGE FOR WOMEN,
VANIYAMBADI
PG & RESEARCH DEPARTMENT OF PHYSICS

CLASS : III BSC PHYSICS
SUBJECT NAME : FUNDAMENTALS OF MICROPROCESSOR-8085
SUBJECT CODE : FEPH 63A

SYLLABUS

UNIT- III

ALP & INSTRUCTION TIMINGS

Assembly language programs-Addition, Subtraction, Multiplication and Division (8-bit only) - Largest/smallest in an array - Sum of series of a se t- T- State - Machine cycle - Instruction cycle-Memory read cycle-Memory write cycle -Wait state - Halt state – Hold state - Delay calculations -Time delay using a single register.

ASSEMBLY LANGUAGE PROGRAMS

The instruction set of 8085 has been discussed elaborately in Chapters 4 and 5. Now we can start learning some assembly language programs. For all the programs, the steps required are given in the form of an algorithm. All the programs can be tried out using microprocessor trainer kits of any make. In this chapter, the user's address range is assumed to be 2000_H to $27FF_H$. The programs can be executed in any other kit with a different address range, by simply changing the higher byte of the address. The programs are given in the following headings.

6.1 Addition

6.2 Subtraction

6.3 Multiplication

6.4 Division

6.5 Square and Square root

6.6 Sorting and Searching

6.7 Code Conversion

6.8 Debugging a program

6.1 ADDITION

a) 8-Bit Addition

(Method 1:-)

Two numbers stored at memory locations 2050_H and 2051_H are added and the result is stored in memory location 2052_H.

Direct addressing mode instructions like LDA... and STA... are used.

Algorithm:

Step 1: Get the first number from the memory location 2050_H to the accumulator.

Step 2: Save the first number in B register.

Step 3: Get the second number from the memory location 2051_H to the accumulator.

Step 4: Add the numbers in A and B.

Step 5: Store the result which is in accumulator in the memory location 2052_H.

Step 6: End.

PROGRAM:

```
LDA     2050H       ;Take the first number to A
MOV     B,A         ;Transfer the number from A into B
LDA     2051H       ;Take the second number to A
ADD     B           ;Add the two numbers in A and B
STA     2052H       ;Store the result in memory
HLT                 ;End of program
```

(Method 2:-)

Two numbers stored at memory locations 2050_H and 2051_H are added and the result is stored in memory location 2052_H.

Register Indirect addressing mode instructions like MOV A,M and MOV M,A are used.

Algorithm:

Step 1: Initialize the memory location 2050_H with HL register.

Step 2: Get the first number from the memory location 2050_H to the accumulator A.

Step 3: Increment the address by one. The memory address is 2051_H.

Step 4: Add the numbers in A and the content of the memory location 2051_H.

Step 5: Increment the address by one. Now the address is 2052_H.

Step 6: Store the result which is in accumulator in the memory location 2052_H.

Step 7: End.

PROGRAM

```
LXI   H,2050H   ;Initialize the memory with HL register
MOV   A,M       ;Transfer the addend from the memory
                ;into the accumulator
INX   H         ;Increment the memory address by one
ADD   M         ;Add the number in A with the augend
INX   H         ;Increment the memory address by one
MOV   M,A       ;Store the result in memory
HLT             ;End of program
```

Example:

Input data:	2050	:	49	(Addend)
	2051	:	85	(Augend)
Result:	2052	:	CE	

b) 8-Bit Addition With Carry

Two numbers stored in memory locations are added and the result is stored in a memory location. After the addition, if the result is greater than FF_H, a carry is produced and a '1' is stored in the next location. If the result is less than FF_H, there is no carry and a '0' is stored.

Algorithm:

Step 1: Get the first number from the memory to the accumulator.

Step 2: Store the first number in B register.

Step 3: Get the second number from the memory to the accumulator.

Step 4: Clear C register to store the carry if the result exceeds FF_H.

Step 5: Add the two numbers.

Step 6: If the result exceeds FF_H, increment the C register by one.

Step 7: Store the sum in memory.

Step 8. Store the carry in memory.

Step 9: End.

PROGRAM:

```
        LDA   2050H      ;Take the first number in A
        MOV   B,A        ;Transfer the first number into B
        LDA   2051H      ;Take the second number in A
        MVI   C,00H      ;Clear the C register
        ADD   B          ;Add the two numbers
        JNC   GOTO       ;If the result is less than FFH,
                         ;Jump to the location named GOTO
        INR   C          ;Else, increment C register
GOTO:   STA   2052H      ;Store the result in memory
        MOV   A,C        ;Move the carry in C to A
        STA   2053H      ;Store the carry in memory
        HLT              ;End of program
```

Example:

Input data: 2050 : 89 (Addend)

2051 : C3 (Augend)

Sum : 2052 : 4C

Carry : 2053 : 01

6.2 SUBTRACTION

a) 8- Bit Subtraction

Two numbers are stored at memory locations 2050_H and 2051_H. The number stored in the memory location 2050_H is subtracted from the number stored in the memory location 2051_H and the result is stored in memory location 2052_H.

Direct addressing mode instructions like LDA... and STA... are used.

Algorithm:

Step 1: Get the first number from memory to the accumulator.

Step 2: Store the first number in B register.

Step 3: Get the second number from memory to the accumulator.

Step 4: Subtract the first number from the second number.

Step 5: Store the difference in the memory.

Step 6: End.

PROGRAM:

```
LDA     2050H       ;Take the first number (subtrahend) in A
MOV     B,A         ;Transfer the first number into B
LDA     2051H       ;Take the second number (minuend) in A
SUB     B           ;Subtract the first number from
                    ;the second number
STA     2052H       ;Store the difference in memory
HLT                 ;End of program
```

Example:

Input data: 2050 : 49 (Subtrahend)
2051 : 85 (Minuend)
Difference: 2052 : 3C

NOTE:

In the example given, we have subtracted a smaller number from a bigger number and obtained a positive result i.e. $85_H - 49_H = 3C_H$. On the other hand, if we subtract the bigger number from the smaller number, we will get a negative result i.e. $(-3C)_H$. This negative result will be stored in 2's complement form in the memory as $C4_H$. In this case, a borrow occurs which is indicated by setting carry flag. In the next program, we get the difference as well as the borrow.

b) 8-Bit Subtraction with Borrow

It is similar to add with carry program. A borrow (carry flag) occurs when a bigger number is subtracted from a smaller number.

Algorithm:

Step 1: Get the first number (subtrahend) from the memory to the accumulator.

Step 2: Store the first number in B register.

Step 3: Get the second number (minuend) from the memory to the accumulator.

Step 4: Clear C register to store the borrow, if the first number is bigger than the second number.

Step 5: Subtract the subtrahend from the minuend.

Step 6: If the first number is bigger than the second number, a borrow is produced which is saved in C register.

Step 7: Store the difference in memory.

Step 8. Store the borrow in memory.

Step 9: End.

PROGRAM:

```
        LDA   2050H      ;Take the subtrahend in A
        MOV   B,A        ;Transfer the subtrahend into B
        LDA   2051H      ;Take the minuend in A
        MVI   C,00H      ;Clear the C register
        SUB   B          ;Subtract the subtrahend from the
                         ;minuend
        JNC   GOTO       ;If there is no borrow, jump
                         ;to the location named GOTO
        INR   C          ;Else, increment C register to
                         ;store the borrow
GOTO:   STA   2052H      ;Store the difference in memory
        MOV   A,C        ;Move the borrow in C to A
        STA   2053H      ;Store the borrow in memory
        HLT              ;End of program
```

6.3 MULTIPLICATION

a) 8-bit multiplication (16 bit result)

In this program, two 8-bit numbers are multiplied using repeated addition method and a 16-bit result is obtained. For example, to multiply $1F_H$ with 26_H, $1F_H$ is added repeatedly 26_H times or 26_H is added $1F_H$ times. The accumulator is initially cleared to use it for the addition procedure. When the partial sum exceeds FF_H, a carry is produced. A counter is initialized with zero and is incremented by one, every time a carry is produced. The accumulator holds the lower byte of the product and the counter register holds the higher byte of the product.

Algorithm:

Step 1: Get the multiplier from the memory to the accumulator.

Step 2: Transfer the multiplier to B register.

Step 3: Get the multiplicand from the memory to the accumulator.

Step 4: Transfer the multiplicand to C register.

Step 5: Initialize accumulator to zero.

Step 6: Clear the D register for storing the carry.

Step 7: Add multiplicand in C register to accumulator.

Step 8: If the value in accumulator exceeds FF_H, increment D register by one.

Step 9: Decrement the multiplier in B by one.

Step 10: If the value in B register is not equal to zero, then go to Step 7.

Step 11: If the value in B register becomes zero. store the lower byte of the result in the accumulator to memory.

Step 12: Store the higher byte of the result in D register in memory.

Step 13: End.

PROGRAM:

```
        LDA 2050H       ;Take the multiplier to accumulator
        MOV B,A         ;Transfer the multiplier to B register
        LDA 2051H       ;Take the multiplicand to accumulator
        MOV C,A         ;Transfer the multiplicand to
                        ;C register
        XRA  A          ;Clear the accumulator
        MOV D,A         ;Clear D register to store the carry
HERE:   ADD C           ;Add the contents of A and C registers
        JNC GOTO        ;If there is no carry, then go to the
                        ;location named GOTO
        INR D           ;Else, increment D register
GOTO:   DCR B           ;Decrement the B register by one
        JNZ HERE        ;If the value in B register is not
                        ;equal to zero, then jump to the
                        ;location named HERE
        STA 2052H       ;Store the lower byte of product in memory
        MOV A,D         ;Transfer the higher byte to accumulator
        STA 2053H       ;Store the higher byte in memory
        HLT             ;End of program
```

6.4 DIVISION

a) 8-Bit Division

Here we divide an 8-bit number by another 8-bit number by repeated subtraction method.

The divisor is subtracted from the dividend repeatedly till the dividend becomes less than the divisor. For each subtraction, a counter is incremented by one. The program generates both the quotient and the remainder.

Algorithm:

Step 1: Get the divisor from the memory to the accumulator.

Step 2: Transfer the divisor to B register.

Step 3: Get the dividend from the memory to the accumulator.

Step 4: Initialize C register with -1 i.e. FF_H to store the quotient.

Step 5: Increment the C register content by one.

Step 6: Subtract the divisor in B from the dividend in A.

Step 7: If the dividend is greater than the divisor, then go to step 5.

Step 8: If the dividend is less than the divisor, add the contents of A and B to get the remainder.

Step 9: Store the remainder in memory.

Step 10: Store the quotient in memory.

Step 11: End.

PROGRAM:

```
        LDA   2050H       ;Take the divisor to accumulator
        MOV   B,A         ;Transfer the divisor to B register
        LDA   2051H       ;Take the dividend to accumulator
        MVI   C,FFH       ;Initialize C register to -1
HERE:   INR   C           ;Increment C register content by one
        SUB   B           ;Subtract the divisor from the
                          ;dividend
        JNC   HERE        ;If the dividend is greater than the
                          ;divisor, then go to the location
                          ;named HERE
        ADD   B           ;Add the contents of A and B to get
                          ;the remainder
        STA   2053H       ;Store the remainder in memory
        MOV   A,C         ;Transfer the quotient in C register
                          ;to accumulator
        STA   2052H       ;Store the quotient in memory
        HLT               ;End of program
```

Example:

Input data:	2050	: 05	(Divisor)
	2051	: 2E	(Dividend)
Result:	2052	: 09	(Quotient)
	2053	: 01	(Remainder)

b) Division by shift and subtract method (16-bit by 8-bit)

Just as multiplication is done by shift and add method, division can be performed by shift and subtract method. The procedure is very much similar to the division we perform with a pen and paper.

We take the most significant eight bits of the dividend and subtract the eight bit divisor. If there is no borrow, the bit of the quotient is set to 1. Otherwise, the quotient is taken as 0. The dividend is shifted left before we do the next subtraction. The dividend and the quotient share a 16-bit register. During each shift, the LSB position falls vacant and in this position, the quotient part is stored. The subtraction is done eight times and a counter is used to keep track of the subtractions.

Algorithm:

Step 1: Load HL register pair with the 16-bit dividend from the memory.
Step 2: Get the 8-bit divisor from the memory into the accumulator.
Step 3: Save the divisor in B register.
Step 4: Load C register with 08_H i.e. C register acts as a counter indicating the number of bits in the divisor.
Step 5: Shift dividend and quotient by one bit position using DAD H.
Step 6: Bring the most significant bits in H to A.
Step 7: Subtract the divisor.
Step 8: If there is a borrow, go to step 10.
Step 9: If there is no borrow, take A back to H and increment L register.
Step 10: Decrement the count in C register by one.
Step 11: If the count is not equal to zero, go to step 5.
Step 12: The H register contains the remainder and L register contains the quotient. Store HL contents in memory.
Step 13: End.

PROGRAM:

```
        LHLD   2050H      ;Load the dividend in HL register
        LDA    2052H      ;Load the divisor in accumulator
        MOV    B,A        ;Save the divisor in B register
        MVI    C,08H      ;Load C register with a count of 08H
LOOP2:  DAD    H          ;Shift HL left once
        MOV    A,H        ;Take the most significant eight bits
                          ;in H to A
        SUB    B          ;Subtract the divisor
        JC     LOOP1      ;If a borrow is produced, go to location
                          ;named LOOP1
        MOV    H,A        ;If no borrow, take the subtracted
                          ;result back to H
        INR    L          ;Put a 1 in LSB position as part
                          ;of the quotient
LOOP1:  DCR    C          ;Decrement the counter
        JNZ    LOOP2      ;If count is not zero, continue
                          ;in LOOP2
        SHLD   2053H      ;Store the remainder and quotient in
                          ;memory
        HLT               ;End
```

Example:

Input data :	2050	: F7	
	2051	: 02	(Dividend)
	2052	: 2C	(Divisor)
Result :	2053	: 11	(Quotient)
	2054	: 0B	(Remainder)

6.6 SORTING AND SEARCHING

a) Largest / smallest number in an array

Let us load N bytes in successive memory locations. The bytes in two successive locations are compared. The larger number is always moved up the array. The smaller number is not destroyed but moved to a lower memory location. If the smaller numbers also are retained, then it will be convenient to modify this program to arrange a set of bytes in ascending order also. When there are N bytes, there will (N-1) comparisons. A register is loaded with (N-1) to act as a counter. After (N-1) comparisons, the largest number is available at the highest memory address.

After the comparison, instead of moving the bigger number up the array, if we move the smaller number, then the same procedure can be used to pick up the smallest number in the array.

The flow chart for picking the largest number in an array is given in Fig (6.2)

Algorithm:

Step 1: Load 'N' bytes in memory whose starting address is in HL register pair.

Step 2: Load (N-1) in C register, to be used as a counter.

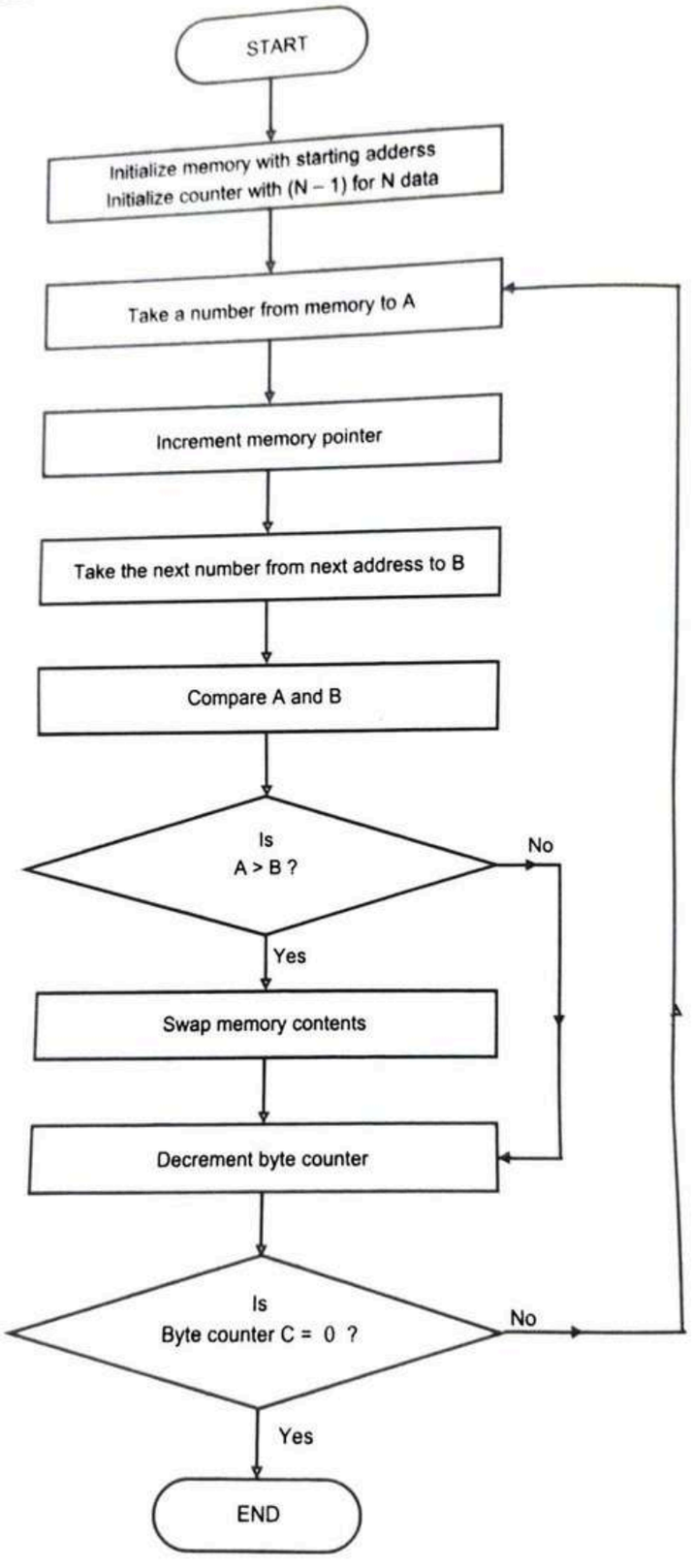

Fig (6.2) Flow chart for Picking up the largest number

Step 3: Move the first data from memory to accumulator.

Step 4: Increment memory address in HL.

Step 5: Move the second data from memory to B register.

Step 6: Compare the data in A register with the data in B register.

Step 7: If data in A register is smaller then go to step 9.

Step 8: Else, 'swap' the first data with the second data stored in memory.

Step 9: Decrement count register C by one.

Step 10: If C register content is not equal to zero, go to step 3.

Step 11: End.

Assuming there are 10 unsigned 8-bit numbers (i.e. the count is $0A_H - 1 = 09_H$) in the array, the program for largest number in an array can be written as follows.

PROGRAM:

```
        LXI    H,2050H     ;Initialize HL register pair with the
                           ;starting address of the memory which
                           ;stores 10 numbers
        MVI    C,09H       ;Load C register with required count
LOOP2:  MOV    A,M         ;Move the first number to accumulator
        INX    H           ;Increment the memory address by one
        MOV    B,M         ;Move the second number to B register
        CMP    B           ;Compare the data in A and B registers
        JC     LOOP1       ;If data in A is smaller, then leave the
                           ;smaller number in its position and
                           ;continue further comparisons
        MOV    M,A         ;Else, 'swap' first data and second
        DCX    H           ;data stored in memory
        MOV    M,B
        INX    H           ;Restore memory address
LOOP1:  DCR    C           ;If comparisons are not over
        JNZ    LOOP2       ;continue in LOOP2
        HLT                ;End of program.
```

Example:

Input data :			Result :		
	2050	: 2A		2050	: 2A
	2051	: FF		2051	: 03
	2052	: 03		2052	: E6
	2053	: E6		2053	: 32
	2054	: 32		2054	: 1C
	2055	: 1C		2055	: D1
	2056	: D1		2056	: 86
	2057	: 86		2057	: A0
	2058	: A0		2058	: 2F
	2059	: 2F		2059	: FF

To pick up the smallest in the array, after the compare instruction, we have to use the instruction JNC instead of JC.

8085 INSTRUCTION TIMINGS

The primary task of a microprocessor is to execute programs. These programs consist of a sequence of instructions stored in the memory. The processor fetches the instructions one by one from the memory and executes them. This operation involves a combination of reading the memory, writing into the memory, reading an input port or writing into the output port. In this Chapter, we will discuss how an instruction is executed by the 8085 and what happens during each clock cycle. This Chapter covers the following topics.

7.1 Introduction
7.2 Memory Read cycle
7.3 Memory Write cycle
7.4 Wait States
7.5 Halt State
7.6 Hold State
7.7 Timing diagrams for some instructions
7.8 Delay calculations

7.1 INTRODUCTION

We have already mentioned that the 8085 microprocessor is connected to a crystal of frequency 6.144 MHz (Chapter 3). The crystal frequency is divided by two internally and the operating frequency of the 8085 system is 3.072 MHz. This gives a time period of 0.325 microseconds. In other words, the system clock frequency is 3.072 MHz and the clock period is 0.325 microseconds. Each step of operation in the processor is synchronized to the clock. The basic operations like memory read, memory write, etc., take three clock periods (or clock pulses). To go little deeper into this chapter, let us define the following terms.

i) T-state is defined as one subdivision of the operation performed in one clock period. These subdivisions are internal states synchronized with the system clock and each T-state is precisely equal to one clock period. The terms T-state and clock period are often used synonymously.

ii) Machine cycle is defined as the time required to complete one operation of accessing memory, I/O, or acknowledging an external request. This cycle may consist of three to six T-states. For some instructions, instruction cycle and machine cycle are the same which means that the instruction takes only one machine cycle. Eg: MOV B,A.

iii) Instruction cycle is defined as the time required to complete the execution of an instruction. The 8085 instruction cycle consists of one to five machine cycles. For example, the instruction MOV B,A takes only one machine cycle whereas the CALL instruction takes five machine cycles.

7.2 MEMORY READ CYCLE

The sequence of operations that takes place when the processor reads a memory location is the memory read cycle. By drawing a timing diagram, we can understand how the address, data and control buses of 8085 behave during the read cycle.

The 8085 processor reads the memory in 3 clock cycles or 3 T-states. That is, 8085 takes 0.925 microseconds to read a memory location. During the 3 T-states, the status of the address, data and control buses can be represented in a timing diagram. The 3 T-states are represented as T_1, T_2 and T_3. The memory read cycle is one of the machine cycles performed by 8085. The timing diagram of memory read cycle is given in Fig (7.1).

The sequence of operations performed by 8085 when it reads a memory location is listed below.

- In the beginning of each machine cycle, i.e., at the beginning of T_1, 8085 sends out a 16-bit address of the memory location that is to be read. The higher order address A_8- A_{15} directly comes out on the higher order address lines A_8- A_{15} and remains unchanged for the entire read cycle. The lower order address A_0- A_7 is sent along the multiplexed address/data lines AD_0- AD_7.
- To demutiplex the multiplexed AD_0-AD_7 lines, 8085 issues the Address Latch Enable (ALE) pulse during T_1. The lower order address A_0- A_7 and data D_0- D_7 are separated using an octal latch and the ALE pulse as explained in Chapter 3, Fig (3.4). The lower order address is separated by the end of T_1. Now the AD_0- AD_7 lines are used for data transfer.

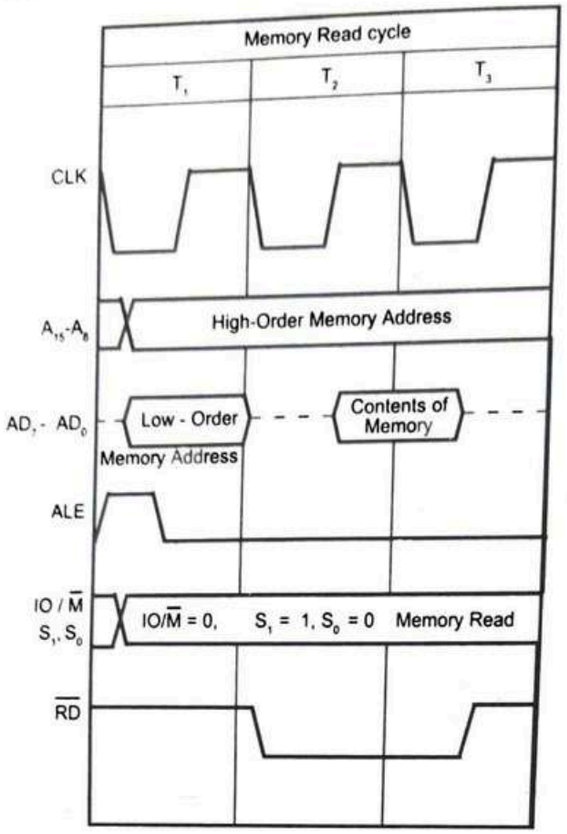

Fig (7.1) Timing Diagram of Memory Read Cycle

- To read the memory, IO/$\overline{M}$ = 0 during T_1 and continues to remain in 0 state till the end of the read cycle.
- Since address and data are separated using the ALE pulse during T_1, the processor makes $\overline{RD}$ = 0, only during T_2, indicating a read operation. The signal $\overline{RD}$ goes back to 1 state in the middle of T_3.
- Combining IO/$\overline{M}$ and $\overline{RD}$ in an OR gate, a memory read signal $\overline{MEMR}$ can be generated and used specifically to read memory devices. Refer Chapter 3, Fig (3.5). The $\overline{MEMR}$ signal is available from the middle of T_2.
- With the address sent by 8085, using external logic circuit, the memory device must be selected. Then, when the $\overline{MEMR}$ signal is applied to the memory device, the contents of the memory location that is addressed is read. The data available on the data lines of the memory chip is taken through the data bus to the processor.

The data bus (AD_0- AD_7) has the memory contents from the middle of T_2 to the middle of T_3. The address A_0- A_7 are separated by the end of T_1. The AD_0- AD_7 lines which now act as the data bus, receive the data only in the middle of T_2. In between, during this small time gap, AD_0- AD_7 lines are undefined. This period is also shown in the timing diagram, given in Fig (7.1). Now the memory read cycle is complete.

7.3 MEMORY WRITE CYCLE

The sequence of operations that takes place when the processor writes a data into a memory is the memory write cycle.

The memory write cycle is identical to the memory read cycle with small changes. The memory write cycle is shown in Fig (7.2).

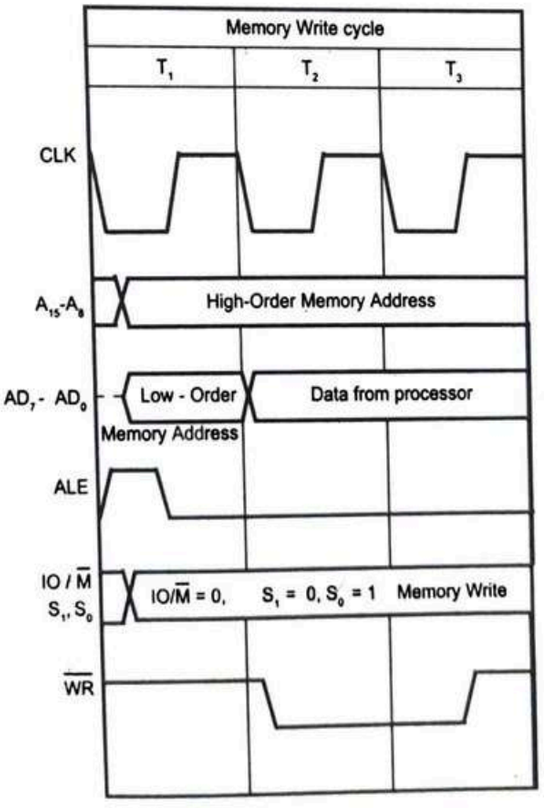

Fig (7.2) Timing Diagram of Memory Write Cycle

Compared to read cycle changes in the write cycle are :

- Since it is a write operation, the processor makes $\overline{WR} = 0$, in the middle of T_2.
- Combining IO/$\overline{M}$ and $\overline{WR}$, a memory write $\overline{MEMW}$ signal can be generated and used to write in a memory device.
- The data to be written into the memory comes from the processor and there is no time lag after the separation of lower order address. The data is placed on AD_0-AD_7 as soon as the address, A_0-A_7 is separated, at the start of T_2 itself.

Note:

The **I/O read machine cycle** is similar to memory read machine cycle. The only change is, IO/$\overline{M}$ = 1 for I/O read machine cycle. Also the **I/O write machine cycle** is similar to memory write machine cycle, again the only change is, IO/$\overline{M}$ = 1.

7.4 WAIT STATES

In 8085, pin 35 is called READY pin. This pin is provided in 8085 to purposely slow down the microprocessor when the processor has to work with slower memory or peripheral devices. If the READY input of the 8085 is made low at the right time, then the processor enters a 'wait state', T_w, after T_2 of the current machine cycle. Fig (7.3a) and (7.3b) shows the timing waveforms for 8085 read machine cycle without and with a wait state respectively. The processor checks the ready input during T_2. If the READY input is high, (Fig 7.3a), the processor proceeds directly to T_3 to complete the read machine cycle in 3 T-states. On the other hand, if the READY input is low during T_2, (Fig 7.3b), then a wait state is introduced. If READY goes high during the wait state, then after the wait state, 8085 continues with T_3 of the machine cycle. With one wait state, the memory read operation takes 4 T-states.

If READY is still low, during the wait state, then one more wait state is introduced. The wait states continue to be inserted as long as READY is low.

During the wait state, the contents of the address bus, data bus and control bus are all held constant.

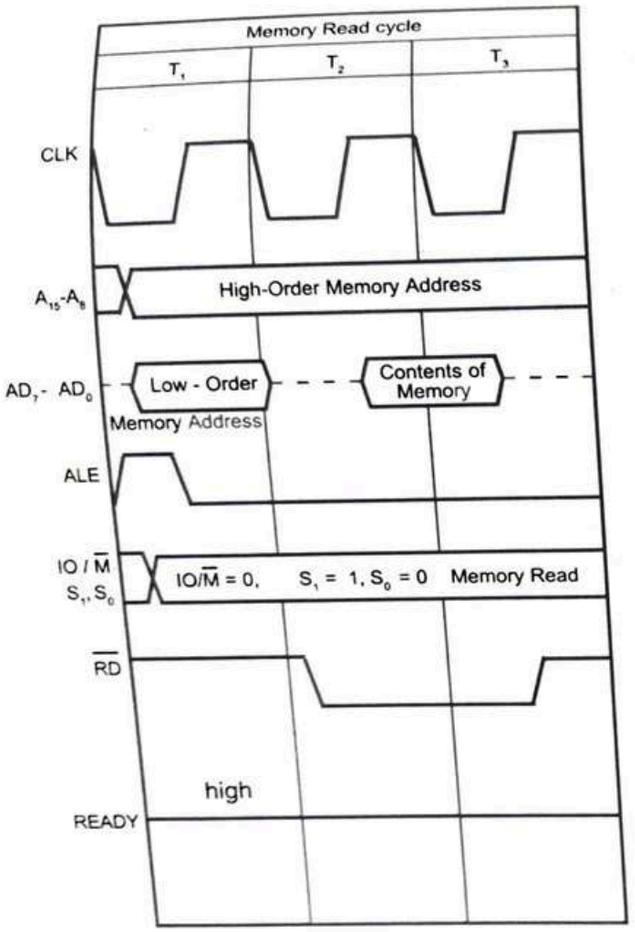

Fig (7.3a) Memory Read Cycle without wait state

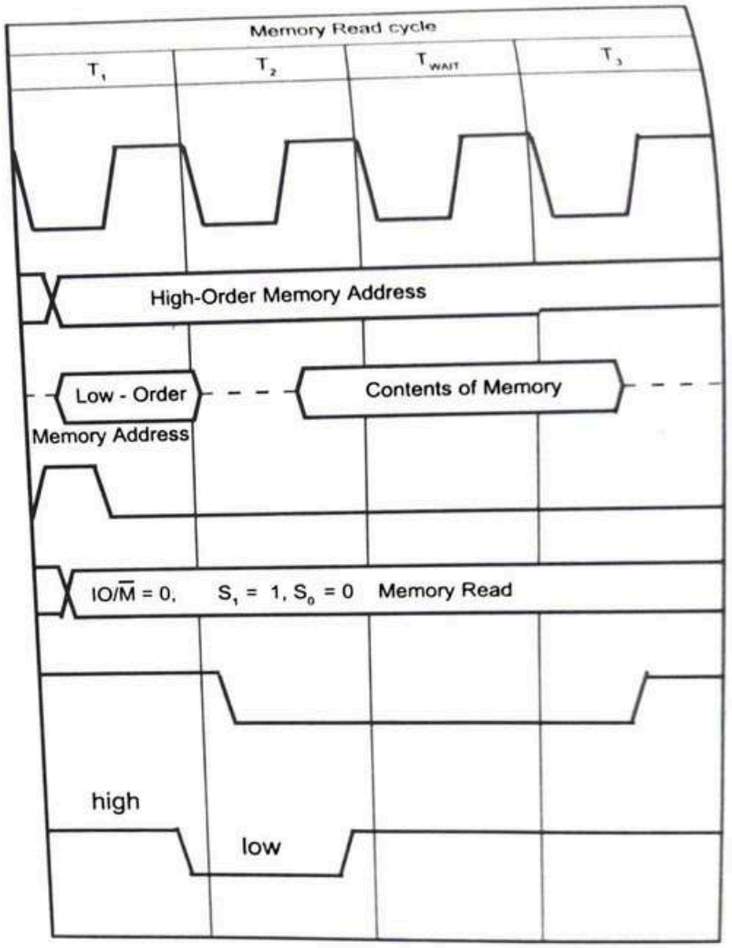

Fig (7.3b) Memory Read Cycle with one wait state

Courtesy Intel ®

The READY pin can be given an active low pulse during T_2 using two D flip-flops connected as shown in Fig (9.4) to insert one wait state.

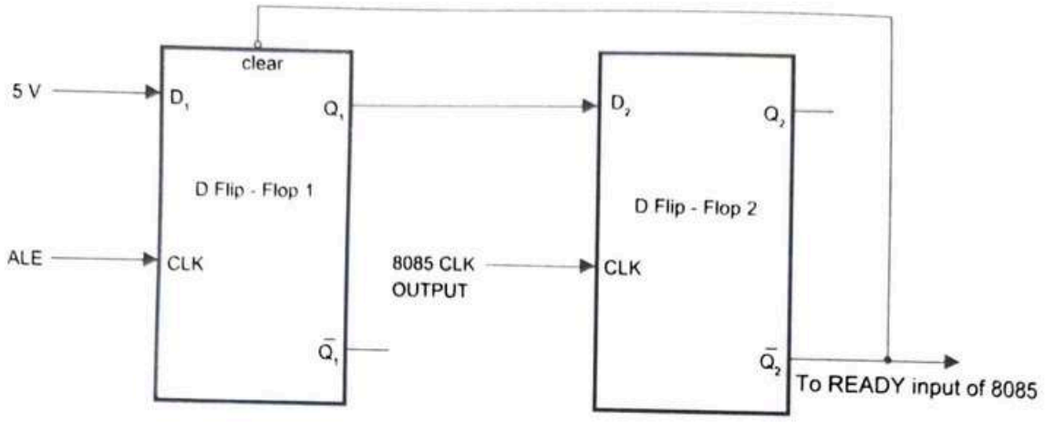

Fig (7.4) Generation of one wait state

The two flip-flops used are positive edge triggered flip-flops (IC 7474). That is, the D input is transferred to the output during the positive edge of the applied clock pulse. The ALE pulse is applied as clock to the first D flip-flop, F/F_1. Since the D_1 input is in 1 state (high), on applying the ALE pulse, the output Q_1 goes high. This Q_1 output is given as input D_2 to the second flip-flop, F/F_2. The clock out of 8085 is used as the clock for the second flip-flop. At the next clock pulse, the output Q_2 now goes high and $\overline{Q}_2$ goes low. This $\overline{Q}_2$ output is connected to the READY pin of 8085 and also to the reset input of the first flip-flop. The READY pin goes low during T_2 and hence 8085 enters a wait state. When $\overline{Q}_2$ goes low, it also resets the first flip-flop and the output Q_1 goes low. Therefore, during T_3 when 8085 is in the wait state, the output Q_2 goes low and $\overline{Q}_2$ goes high. The READY pin also goes high and 8085 enters T_3 after only one wait state. Since the ALE pulse is used, one wait state will be introduced for each machine cycle.

A monostable multivibrator, which produces a single pulse of finite duration, can also be used to give a low pulse to the READY input. The $\overline{RD}$ or $\overline{WR}$ signal may be used to trigger the monostable multivibrator to introduce a wait state for each read or write machine cycle. If wait state is to be inserted only for a particular device that is connected to 8085, then the chip select or chip enable signal developed during the interface can be used to trigger the monostable multivibrator. This avoids wait states being introduced for each machine cycle.

7.5 HALT STATE

When the halt instruction, HLT is executed, the 8085 will enter a halt state after T_2 of the next machine cycle. In the halt state, the processor is stopped; the buses are tristated. There are only three ways to get out of the halt state;

1. Reset 8085 by applying an active low input to RESETIN of 8085.
2. Make HOLD pin high so that 8085 enters a hold state. But, when the hold input goes low again, the processor returns to the halt state.
3. Interrupt 8085, by applying an active high signal to one of the interrupt pins. (8085 - Interrupts is discussed in Chapter 10).

7.6 HOLD STATE

In 8085, pin 39 is called the HOLD pin. The HOLD line is checked by the processor during the HALT state and T_2 state of a machine cycle. If the HOLD pin is high, the microprocessor enters a HOLD state and issues a 'hold acknowledge' signal through the HLDA pin (pin 38). After recognizing an active high HOLD signal, the processor suspends the execution of further machine cycles. The address bus, data bus and control bus go into high impedance. During the HOLD state, peripheral devices can make use of the three buses to transfer data to or from the memory, directly. This mode of operation is known as Direct Memory Access (DMA). A brief explanation of DMA is given in Chapter 12.

7.7 TIMING DIAGRAMS **FOR** SOME INSTRUCTIONS

We have mentioned about execution of an instruction by a microcomputer in general, in Chapter 2, Sec 2.9. In this section, we will discuss how the microprocessor 8085 executes different types of instructions. All instructions in 8085 are executed using some combination of the four machine cycles, namely, i) Memory Read ii) Memory Write iii) I/O Read iv) I/O Write

For all the instructions, the first machine cycle is always a memory read cycle, fetching the opcode from the memory. Therefore, the first machine cycle is called as opcode fetch machine cycle. The memory read, memory write, I/O read and I/O write machine cycles take 3 clock cycles or 3 T-states. But, the opcode fetch machine cycle in 8085 takes 4 T-states or 6 T-states. Let us see how some instructions are executed and draw the diagrams for these instructions.

a) ***Timing diagram for MOV B,A instruction:***

Let us assume that the machine code of the instruction is stored at location 2015_H. The memory address and its contents are as shown below.

Instruction	Mem Addr.	Hex code
MOV B,A	2015	47

Let us also assume that the program counter is ready with the address 2015_H. This instruction is completed in one machine cycle itself. The first machine cycle fetches the hexcode of the instruction from the memory and this cycle of operation is called opcode fetch. For the instruction MOV B,A , 4 T-states are required. The timing diagram is given in Fig (7.5).

The opcode fetch cycle is similar to the memory read cycle that we have seen in Sec 7.1. The memory read takes 3 T-states, where as, for the instruction MOV B,A the opcode fetch takes 4 T-states. The sequence of execution of the instruction is as follows.

The 8085 places the contents of the program counter on the address bus. The higher byte 20_H is placed on A_8- A_{15}. The lower byte 15_H is placed on AD_0-AD_7 which is used to carry the address A_0- A_7 only during T_1. The higher address lines remain unchanged for the first three T-states, T_1, T_2 and T_3. During T_4, the higher order address lines are undefined. The lower address A_0-A_7 are separated using an address latch and the ALE pulse. For this purpose, 8085 issues ALE pulse during T_1 itself. The ALE pulse goes high during T_1 but goes back to low state and remains in low state for the remaining three T-states. Once the lower order address is separated, the lines AD_0- AD_7 are used to transmit data or in other words, AD_0- AD_7 becomes the data bus. In initial period of T_2, till it receives the data, the data bus is in undefined state.

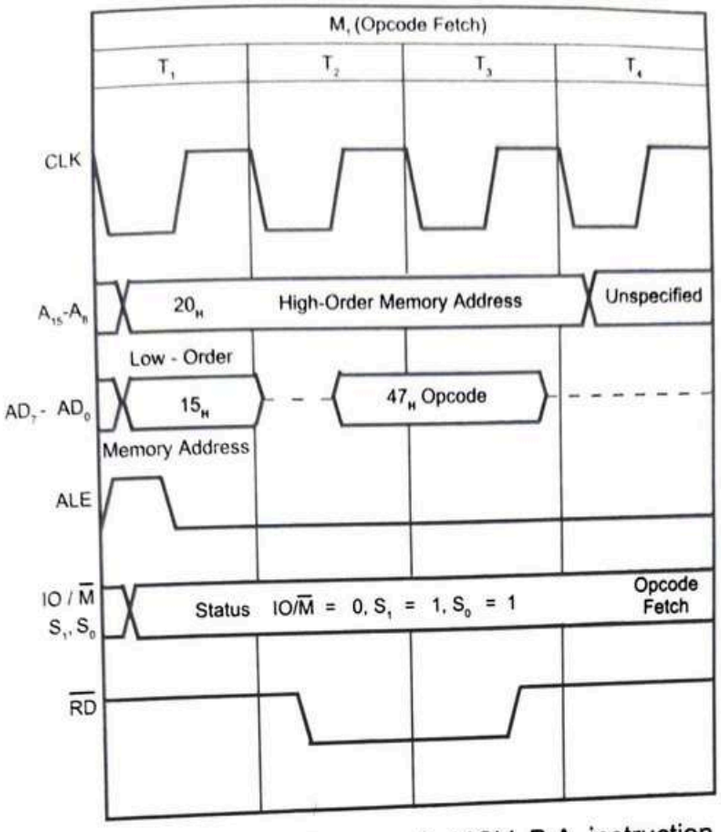

Fig (7.5) Timing Diagram of MOV B,A instruction

Since the opcode fetch operation is a memory read operation, 8085 makes the signal $IO/\overline{M}$ equal to zero. The status signals S_1S_0 reads 11 indicating opcode fetch operation. $IO/\overline{M} = 0$ and $S_1S_0 = 11$ remain unchanged for the entire machine cycle.

The 8085 makes the $\overline{RD}$ signal low in the middle of T_2 to read the memory. For the given address 2015_H, the memory is selected and with $\overline{RD}$ signal, the contents of the memory location, 47_H is read. This hexcode falls on the data bus in the middle of T_2.

The 8085 now reads the hexcode and puts it in the instruction register. After reading the machine code 47_H, $\overline{RD}$ signal is made high towards the end of T_3.

In the fourth T-state, T_4, the instruction is decoded and the relevant operation is carried out. In this case, the contents of A register is copied in B register. Now, the instruction cycle is completed in one machine cycle, taking 4 clock cycles. Or, the time taken to execute the instruction MOV B,A is 4T (4 x 0.325 microseconds).

MOV B,A is a one byte instruction involving 8-bit registers. A number of other one byte instructions like MOV C,D, ADD B, INR D etc., also have only one machine cycle and the instructions are completed in 4 clock periods.

b) *Timing diagram for DCX D instruction;*

This instruction is also a one byte instruction and it is also executed in one machine cycle. But, since the operation involves the register pair DE, 8085 takes 6 clock cycles to complete the instruction. The timing diagram for the instruction DCX D is given in Fig (7.6). As in the previous example, let us assume the instruction is loaded in at the address 2015_H

Instruction	Mem Addr	Hexcode
DCX D	2015	1B

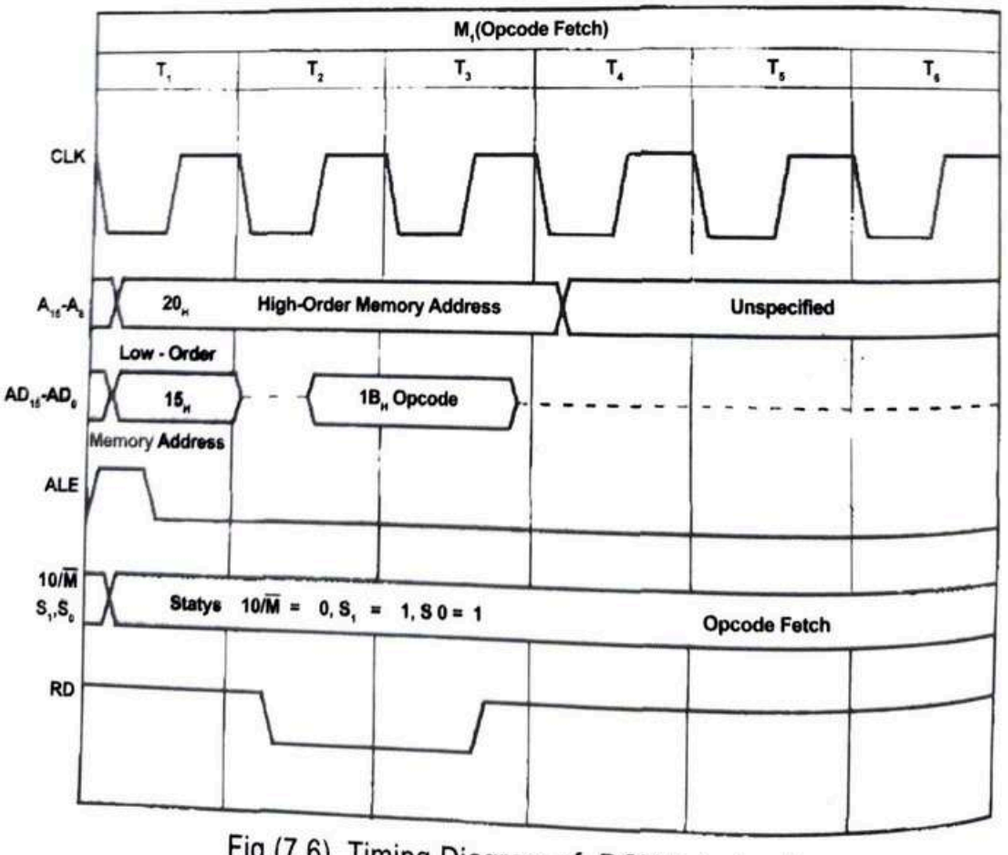

Fig (7.6) Timing Diagram of DCX D instruction

We can see that for the first 3 T-states, the timing diagram for MOV B,A and DCX D are the same. The instruction MOV B,A, after fetching the machine code completes the operation in the fourth T-state. The instruction DCX D completes the operation of decrementing the register pair DE by 1 in three clock cycles T_4, T_5 and T_6.

During T_4, T_5 and T_6, the higher order address bus A_8-A_{15} and the data bus D_0-D_7 are undefined. The signals $IO/\overline{M} = 0$ and $S_1S_0 = 11$ continue as it is for the entire machine cycle of 6 T-states.

The ALE continues to remain low and the $\overline{RD}$ continues to remain high.

The instruction DCX D is completed in 6 T-states.

c) *Timing Diagram for MVI A,25$_H$ instruction:*

This is a two byte instruction. When executed, this instruction moves the 8-bit data that follows the opcode into the accumulator. In other words, the data to be moved to the accumulator is available in the very next memory location following the machine code of the instruction. This instruction is executed in two machine cycles and takes 7 T-states. The first machine cycle is opcode fetch and the second machine cycle is memory read. Let us assume the instruction is loaded in at the address 2015_H.

Instruction	Mem Addr	Hexcode
MVI A,25$_H$	2015	3E
	2016	25

Opcode Fetch (4T): This machine cycle is completed in 4 T-states. The explanations are almost the same as the opcode fetch machine cycle for our first example, MOV B,A instruction. The sequence of operations are once again repeated.

The 8085 places the contents of the program counter on the address bus. The higher byte 20_H is placed on A_8-A_{15}. The lower byte 15_H is placed on AD_0-AD_7 which is used to carry the address A_0-A_7 only during T_1. The higher address lines remain unchanged for the first three T-states T_1, T_2 and T_3. During T_4, the higher order address lines are undefined.

We can see that for the first 3 T-states, the timing diagram for MOV B,A and DCX D are the same. The instruction MOV B,A, after fetching the machine code completes the operation in the fourth T-state. The instruction DCX D completes the operation of decrementing the register pair DE by 1 in three clock cycles T_4, T_5 and T_6.

During T_4, T_5 and T_6, the higher order address bus A_8-A_{15} and the data bus D_0-D_7 are undefined. The signals $IO/\overline{M} = 0$ and $S_1S_0 = 11$ continue as it is for the entire machine cycle of 6 T-states.

The ALE continues to remain low and the $\overline{RD}$ continues to remain high.

The instruction DCX D is completed in 6 T-states.

c) *Timing Diagram for MVI A,25$_H$ instruction:*

This is a two byte instruction. When executed, this instruction moves the 8-bit data that follows the opcode into the accumulator. In other words, the data to be moved to the accumulator is available in the very next memory location following the machine code of the instruction. This instruction is executed in two machine cycles and takes 7 T-states. The first machine cycle is opcode fetch and the second machine cycle is memory read. Let us assume the instruction is loaded in at the address 2015_H.

Instruction	Mem Addr	Hexcode
MVI A,25$_H$	2015	3E
	2016	25

Opcode Fetch (4T): This machine cycle is completed in 4 T-states. The explanations are almost the same as the opcode fetch machine cycle for our first example, MOV B,A instruction. The sequence of operations are once again repeated.

The 8085 places the contents of the program counter on the address bus. The higher byte 20_H is placed on A_8-A_{15}. The lower byte 15_H is placed on AD_0-AD_7 which is used to carry the address A_0-A_7 only during T_1. The higher address lines remain unchanged for the first three T-states T_1, T_2 and T_3. During T_4, the higher order address lines are undefined.

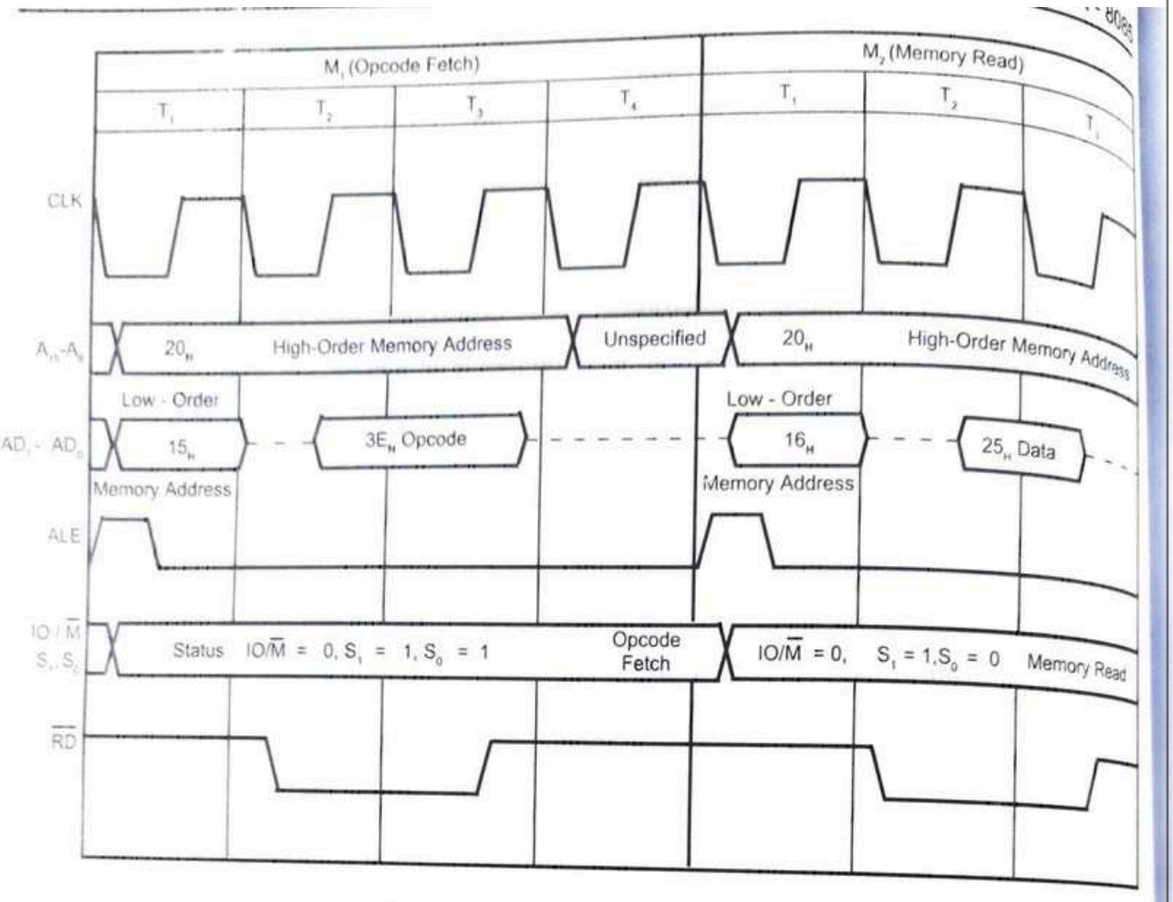

Fig (7.7) Timing Diagram of MVI A,25_H instruction

The lower address A_0-A_7 are separated using an address latch and the ALE pulse. For this purpose, 8085 issues ALE pulse during T_1 itself. The ALE pulse goes high during T_1 but goes back to low state and remains in low state for the remaining three T-states. Once the lower order address is separated, the lines AD_0-AD_7 are used to transmit data or in other words, AD_0-AD_7 becomes the data bus. In initial period of T_2, till it receives the data, the data bus is in undefined state.

Since the opcode fetch operation is a memory read operation, 8085 makes the signal $IO/\overline{M}$ equal to zero. The status signals S_1S_0 read 11 indicating opcode fetch operation. $IO/\overline{M} = 0$ and $S_1S_0 = 11$ remain unchanged for the entire machine cycle.

The 8085 makes the $\overline{RD}$ signal low in the middle of T_2 to read the memory. For the given address 2015_H, the memory is selected and with $\overline{RD}$ signal, the contents of the memory location, 2015_H i.e. $3E_H$ is read. This hexcode falls on the data bus in the middle of T_2.

The 8085 now reads the hexcode and puts in the instruction register. After reading the machine code $3E_H$, $\overline{RD}$ signal is made high towards the end of T_3.

In the fourth T-state, T_4, the instruction is decoded and the processor decides that one more machine cycle is to be performed to read the data from the memory and transfer it to the accumulator. The machine cycle is the memory read machine cycle.

Memory Read (3T): The memory read machine cycle is the same as the memory read cycle we have explained in Chapter 7, Sec.7.1. Therefore, we can say that 8085 sends the address 2016_H over the address bus, makes IO/$\overline{M}$ = 0, $\overline{RD}$ = 0 and reads the contents of the memory location 2016_H. The contents of this location which is 25_H is transferred to the processor through the data bus. Within the 3 T-states, the data 25_H is placed in the accumulator.

The timing diagram for the instruction MVI A,25_H is given in Fig (7.7).

d) *Timing Diagram for LXI H,2050_H instruction:*

This is a three byte instruction. When executed, this instruction transfers the 16-bit data that follows the opcode into HL register pair. The lower byte, 50_H is transferred to the L register and the higher byte 20_H is transferred to the H register. This instruction is executed in three machine cycles and takes 10 T-states. The first machine cycle is opcode fetch (4T) and the other two are memory read machine cycles (3T + 3T). Let us assume the instruction is loaded in at the address 2015_H.

Instruction	Mem.Addr	Hexcode
LXI H,2050_H	2015	21
	2016	50
	2017	20

In opcode fetch cycle, the 8085 fetches the machine code of the instruction 21_H from the memory location 2015_H. In the fourth clock cycle of this opcode fetch machine cycle, the processor decodes the instruction and decides that it has to perform two more memory read cycles to read two bytes from successive locations and load them in HL registers.

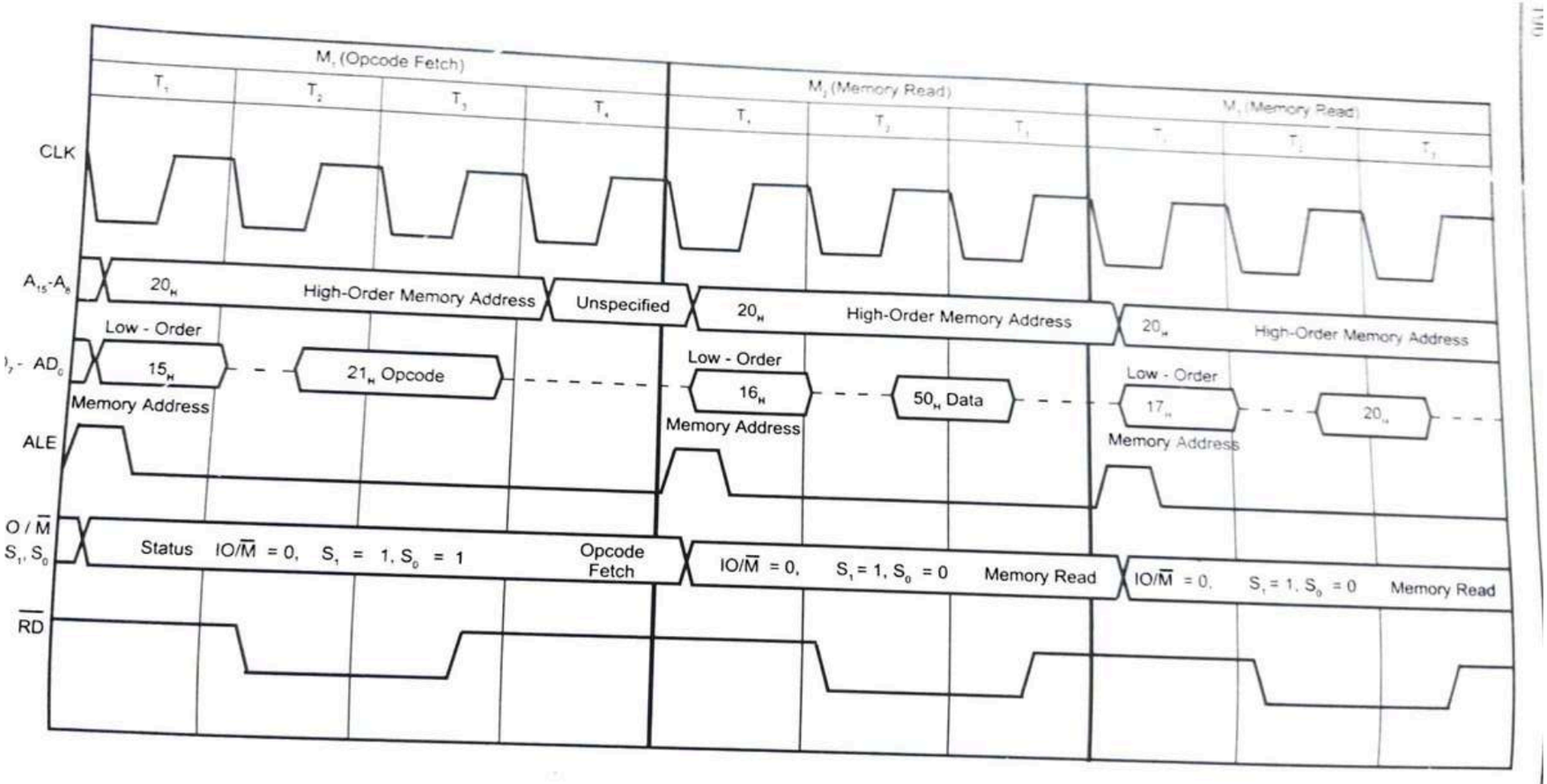

Fig (7.8) Timing Diagram of LXI H,2050_H instruction

In the first memory read cycle, the processor sends the address 2016_H and reads the data 50_H which is transferred to L register.

In the second memory read cycle, the processor sends the address 2017_H and reads the data 20_H which is transferred to H register.

We should remember that in 8085, when a 16-bit number is loaded in the memory, the lower byte is loaded in the memory location having the lower address and the higher byte is loaded in the memory location having the higher address.

The timing diagram for the instruction is given in Fig (7.8). The instruction LXI H,2050_H is completed in 3 machine cycles and takes 10 T-states.

e) ***Timing Diagram for STA 2275_H instruction:***

This is a three byte instruction. When executed, this instruction transfers the contents of the accumulator to a memory location whose address is 2275_H which is directly given in the instruction itself. This instruction is executed in four machine cycles and takes 13 T-states. The first machine cycle is opcode fetch (4T), the second and third are memory read machine cycles (3T + 3T) and the fourth one is a memory write machine cycle (3T). Let us assume the instruction is loaded in at the address 2015_H.

Instruction	Mem.Addr	Hexcode
STA 2275_H	2015	32
	2016	75
	2017	22

In opcode fetch cycle, the 8085 fetches the machine code of the instruction 32_H from the memory location 2015_H. In the fourth clock cycle of this opcode fetch machine cycle, the processor decodes the instruction and decides that it has to perform two more memory read cycles, to read two bytes from successive locations and get the address 2275_H. After that, the processor performs a memory write operation as the fourth machine cycle to write the contents of the accumulator in the memory location whose address is 2275_H.

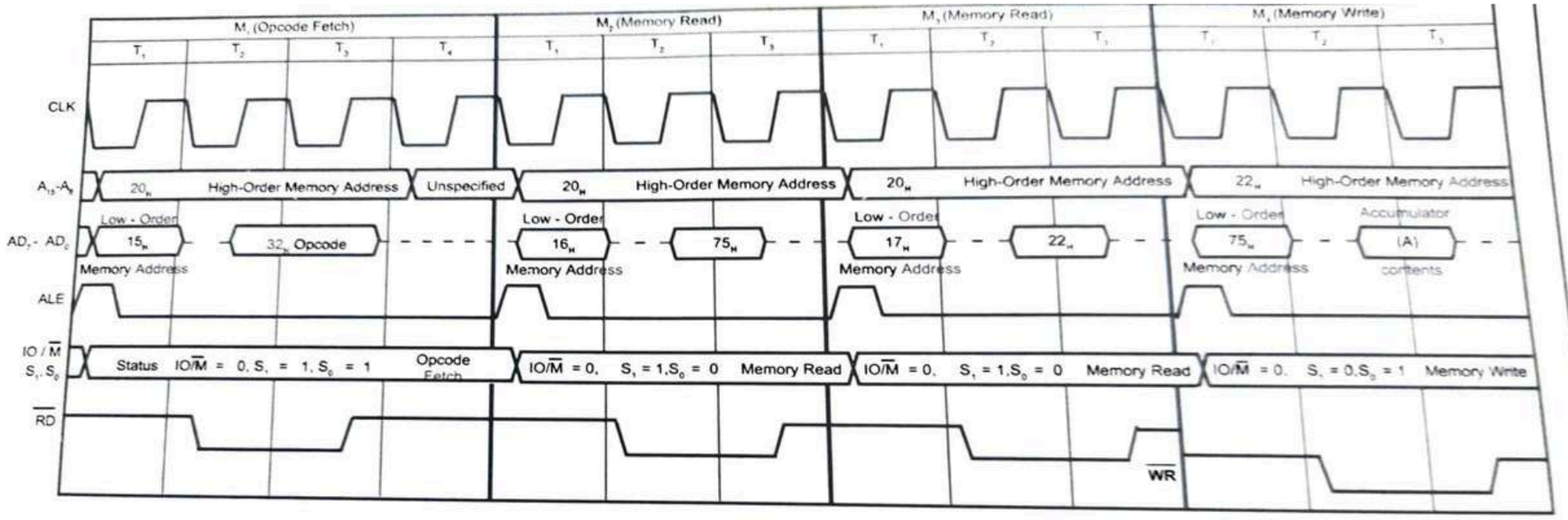

Fig (7.9) Timing Diagram of STA 2275_H instruction

In the first memory read cycle, the processor sends the address 2016_H and reads the data 75_H. This is the lower byte of the address which is stored temporarily inside the processor.

In the second memory read cycle, the processor sends the address 2017_H and reads the data 22_H. This is the higher byte of the address which is also stored temporarily inside the processor.

The fourth machine cycle is the memory write cycle. The processor sends the higher byte of the address 22_H over the lines A_8 - A_{15} and the lower order address 75_H over the lines AD_0 - AD_7. As explained earlier, the lower order address A_0 - A_7 is separated from the multiplexed AD_0 - AD_7 lines using the ALE pulse and an external latch. Since it is a write operation, IO/$\overline{M}$ becomes 0 during T_1 itself and $\overline{WR}$ becomes 0 during T_2. The processor places the accumulator contents on the data bus during T_2. Now the address 2275_H is given to the memory through the address bus. The contents of the accumulator are placed on the data bus. Also, a memory write signal is given to the memory. The contents of the accumulator are now written into the memory whose address is 2275_H.

Now the instruction STA 2275_H is completed in four machine cycles. The timing diagram for the instruction STA 2275_H is given in Fig (7.9).

7.8 DELAY CALCULATIONS

From the above discussions, it is clear that each instruction is completed in finite number of machine cycles and hence in a finite number of clock cycles. If we know the number of clock cycles for an instruction and the clock period, then we can calculate the time taken to execute that instruction. The number of T-states required for each instruction along with their hex-codes is given in Appendix-I.

Once we know how to calculate the time taken to execute a particular instruction, then it is possible to calculate the total time taken to complete a group of instructions. This is used to write precise 'time delay' routines. Time delay routines can be used to produce delays of a few milliseconds to a few minutes or even few hours. Delay programs are used in

programs for waveform generation, stepper motor interface, flashing LEDs, 12 / 24 hour clock simulation, data acquisition systems, etc.,

Assuming the time delay program is written as a subroutine, let us see how the routine is written and calculate the delay involved.

a) Time Delay using a Single Register

In this method, a single register is loaded with a desired number. The content of the register is decremented one by one. The program moves in a loop till the content of the register becomes zero. Then the program returns to the main program with a RET instruction.

Let us load C register with FF_H and calculate the delay involved. The delay subroutine and the T-states for each instruction is given below.

```
        MVI   C,FFH      7T
LOOP    DCR   C          4T
        JNZ   LOOP       7T / 10T
        RET              10T
```

The instruction JNZ takes 10 T-states if the content of C register is not zero and the program jumps back to the label LOOP. When the content of C register becomes zero, the program continues in its regular sequence to execute the RET instruction and for this, JNZ takes only 7 T-states.

Let us represent JNZ with 10 T-states as JNZ_{10} and JNZ with 7 T-states as JNZ_7.

The instructions MVI C, FF_H and RET are executed only once. DCR C is executed 255 times. JNZ is also executed 255 times, but for 254 times it takes 10 T-states and for once, it takes 7 T-states. Therefore, the total time delay involved can be written as

Delay in secs = Delay outside the loop + Delay inside the loop

Delay outside the loop = Time to execute (MVI C, FF_H + DCR C + JNZ_7 + RET) once.

Therefore, if the count loaded in C register is N in decimal, then in general, the time delay can be calculated as

Delay outside the loop = Time to execute (MVI C, N + DCR C + JNZ, + RET) once.

Delay inside the loop = Time to execute (DCR C + JNZ_{10}) (N – 1) times.

Delay outside the loop = 7 T + 4 T + 7 T +10 T = 28 T

Delay inside the loop = (4 T + 10 T) (N – 1)

Therefore,

Total delay in secs = [28 + 14 (N – 1)] T

where T = 0.325 microsecs = 325 nanosecs = 325 ns, when the crystal frequency is 6.144 MHz.

For N = $FF_H = 255_{10}$

Delay = [28 + 14 X 254] 325 X 10^{-9} secs

= [28 + 3556] 325 X 10^{-9} secs

= 3584 X 325 X 10^{-9} secs

= 1164800 X 10^{-9} secs

= 1.1648 X 10^{-3} secs

= 1.1648 msecs

NOTE:

Since the delay is called from the main program, the time taken for the CALL instruction can also be included in the delay calculation, if we want to be very precise. CALL instruction takes 18 T-states.

Therefore, the delay outside the loop is 28 T + 18 T = 46 T-states.

If we want to increase the delay without modifying the program, we can introduce NOP instruction one or more times which increases the T-states within the loop. The modified program with one NOP is shown below.

	MVI	C, FF_H	7T
LOOP:	DCR	C	4T
	NOP		4T
	JNZ	LOOP	10T / 7T
	RET		10T

Now, Delay = [32 + 14 (N – 1)] T without CALL instruction

Delay = [50 + 14 (N – 1)] T with CALL instruction.

b) Delay using two registers

In the first program, a single register C was loaded with the count FF_H and decremented one by one till the count reaches zero. This part of the program can be kept in an inner loop and can be repeated number of times using another register to generate larger delay. Let us take FF_H in C register and $0A_H$ in B register and repeat the inner loop $0A_H$ times or ten times. The program is given below.

```
        MVI   B,0AH     ; Load 10 in B register
LOOP 1: MVI   C,FFH     ; Load 255 in C register
LOOP 2: DCR   C         ; Decrement count in C register by one
        JNZ   LOOP 2    ; If (C) is not equal to zero, continue in LOOP 2
        DCR   B         ; Decrement count in B register by one
        JNZ   LOOP 1    ; If (B) is not equal to zero, continue in LOOP 1
        RET             ; Return to the main program
```

By changing the count in B and C registers, time delay of different magnitudes can be generated.

c) Time delay using register pair

Instead of using two registers B and C separately as shown above, the register pair BC (or DE or HL) can be loaded with a 16-bit number and decremented one by one till the count reaches zero. This will be very useful in generating delays of 1 second or more. The point to remember is that the instruction that decrements the register pair (DCX B, etc..)

does not affect the zero flag. Therefore, we have to think of another way to determine whether the content of the register pair has reached zero after the count down. The content of the lower register (or higher register) is brought to the accumulator and logical OR operation is done with the content of the higher register (or lower register). The zero flag will be set by the ORA instruction only when the contents of both the registers are 0's.

Let us write a program using the register pair DE, loaded with $FFFF_H$.

```
        LXI   D,FFFF_H    ; Load 65,535 in DE register pair
LOOP:   DCX   D           ; Decrement the count in DE by one
        MOV   A,D         ; Copy the content of D in A
        ORA,  E           ; OR the contents of A and E
        JNZ   LOOP        ; If zero flag is not set, continue in LOOP
        RET               ; If count has reached zero, return to the main
                          ; program
```

Knowing the T-states for each instruction, the delay generated by the above program can be calculated.

Instruction	T-states
LXI D	10
DCX D	6
MOV A,D	4
ORA, E	4
JNZ	7/10
RET	10

If the count in DE register pair is N,

the total number of T-states

= T-states of (LXI D.. + (DCX D + MOV A,D + ORA E)N + JNZ_{10}(N-1) + JNZ_7 + RET)

T-states

= 10 + (6+4+4)N + 10(N-1) + 7 + 10

= 24N + 17

Time delay = (24N + 17) X T seconds

Omitting 17, time delay = 24 X 65,535 X 0.325 X 10^{-6} = 0.511 sec.

Practice Questions:

1. What is the purpose of the MOV instruction in 8085 assembly language?
2. What is the difference between the ADD and SUB instructions in 8085?
3. Explain how you would perform multiplication of two 8-bit numbers in 8085 assembly language.
4. What is the function of the HLT instruction in an assembly program?
5. In the context of the 8085, what is the role of the accumulator in arithmetic operations?
6. What is the purpose of using MOV and MVI instructions in data transfer?
7. What is the significance of the JZ instruction in 8085 assembly language?
8. What does the CMP instruction do in the 8085 microprocessor?
9. What is the purpose of a time delay in an assembly language program?
10. What is the basic function of the CALL and RET instructions in 8085 assembly language?

Additional Resources :

https://www.geeksforgeeks.org/instruction-cycle-8085-microprocessor/

https://www.shivajicollege.ac.in/sPanel/uploads/econtent/dd451b1ee29870bdfd26b6ecaa84de98.pdf

References:

1. Fundamentals of Microprocessor 8085-V.Vijayedran