Block dig. of 8085 consists of the following units -

1. Arithmetic and logic unit
2. Registers / Register array - GPR, A, Temp. reg.
3. Instruction register & decoder
4. Interrupt control
5. Serial I/O control
6. Timing and Control unit
7. 16-bit Registers

(1) ALU - µp has 8-bit ALU, it performs arithmetic operations such as 8-bit binary add, sub, 8/16 bit binary addn, 8 bit BCD add, inc/dec of 8/16 bit data & logical operations such as ANDing, ORing, EX-ORing of 2 8 bit nos, inverting, comparison & rotating 8 bit nos towards left/right, with or without carry.

It includes :- Accumulator (result is stored here)
- Temporary register (to hold data during ALU operation)
- ALU ckts.
- Five flags (set or reset acc. to the result of operation)

Flag register or PSW (Program Status Word) -
It is 8-bit register which contains 8 ff's. Bits D5, D3 & D1 are not used, rest of five flip-flops are called as status flags

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

(1) Sign Flag (S) - S=1 if D_7 of result (usually in A) is 1 i.e. no is negative
S=0, then result is positive (D_7 of result 0)
($+127_D$ to -128_D)
→ used with signed nos where D_7 indicate sign by removing 7 bits represent magnitude
→ In case of unsigned nos (illegible)
flag

i) Zero flag (Z) - Z = 1 if ALU operation results in 0
= 0 " " " " is not 0.
- It is modified by result in acc as well as other registers

iii) Auxiliary carry (AC) -
- AC = 1 when carry is generated by D_3 & passed on to bit D_4 (i.e. it indicates status of 4 LSBs)
- AC = 0, otherwise
- used for BCD operations & is not available for the programmer to change seq. of program using jump instruction

iv) Parity flag (P) - It is the no of 1's in the result.
P = 1 if result has even no of 1's (even parity)
P = 0 " " " odd no of 1's (odd parity)

v) Carry flag (CY) - When we add 2 8/16 bit nos, max. bits in the result can be 9/17. 9th MSB is copied in to carry flag.
- CY = 1 when 9th bit is 1
- CY = 0 " " " " 0.

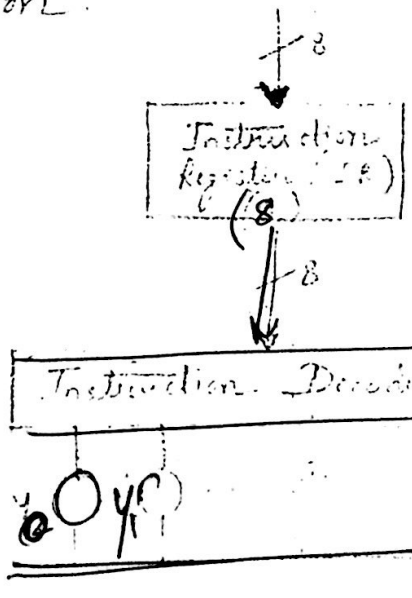
∴, CY flag also serves as a borrow flag for subtraction
- In case of subtraction,
CY = 0 if x > y (borrow is not required)
CY = 1 if x < y (" " required)

(2.) Register Array - → (General purpose registers)
(i) There are six 8-bit registers :- B, C, D, E, H and L.
(ii) For storing data greater than 8 bit, these registers can be used in pairs - BC, DE, HL (Each pair can store 2 bytes of data)

→ Accumulator (A) – It is also a 8-bit register
– In most of the arithmetic & logical op. µp always take first 8 bit no. from A & 8 LSB's of result in ALU are stored back to A.

→ Temporary Registers – Registers W & Z are included in the register array. They are used to hold 8-bit data during execution of some instructions.
They are used internally & not available to the programmer.

3) Instruction Register & Decoder –
– They are the part of ALU.
– When an instruction is fetched from memory, it is loaded in the Instruction register. Decoder decodes the instruction & establishes the sequence of events to follow.
– IR is not programmable & cannot be accessed through any instruction.



∴ ID has 8 i/ps, there are $2^8 = 256$ o/ps. Each o/p is connected to the controlling ckt. µp can perform 256 diff operations

Instruction register (IR) and

(4) Interrupt Control - It is responsible for enabling & disabling of interrupts

There are 5 h/w interrupts - INTR, RST 5.5, RST 6.5, RST 7.5, TRAP

(5) Serial I/O Control

with 8085 parallel data transfer is possible through D_0-D_7 but to send data serially SOD & SID pins of µp can be used.

(6) Timing and Control Unit

This unit synchronizes all µP operations with clock and generates the control signals necessary for comm. b/w µp and peripherals. RD & WR are sync pulses indicating availability of data on the data bus

(7) 16-bit registers

i) Program Counter (PC) - (or Memory pointer)

- It acts as a pointer to the next instruction to be executed & contains 16-bit address of memory location of next instruction
- It is of 16-bit as memory locations have 16-bit addresses
- It points to the memory address from which next byte is to be fetched. When a byte is fetched, PC is incremented by one to point to the next memory location

ii) Stack pointer (SP) -

- It points to a memory location in R/w memory (RAM) called as stack; where info. is stored on first in last out (FILO) basis
- Beginning of stack is defined by loading 16-bit address in the stack pointer (top of stack)
- After data is written onto stack, SP holds address of last byte written onto the stack

Configuration / Pin dig. of 8085 –

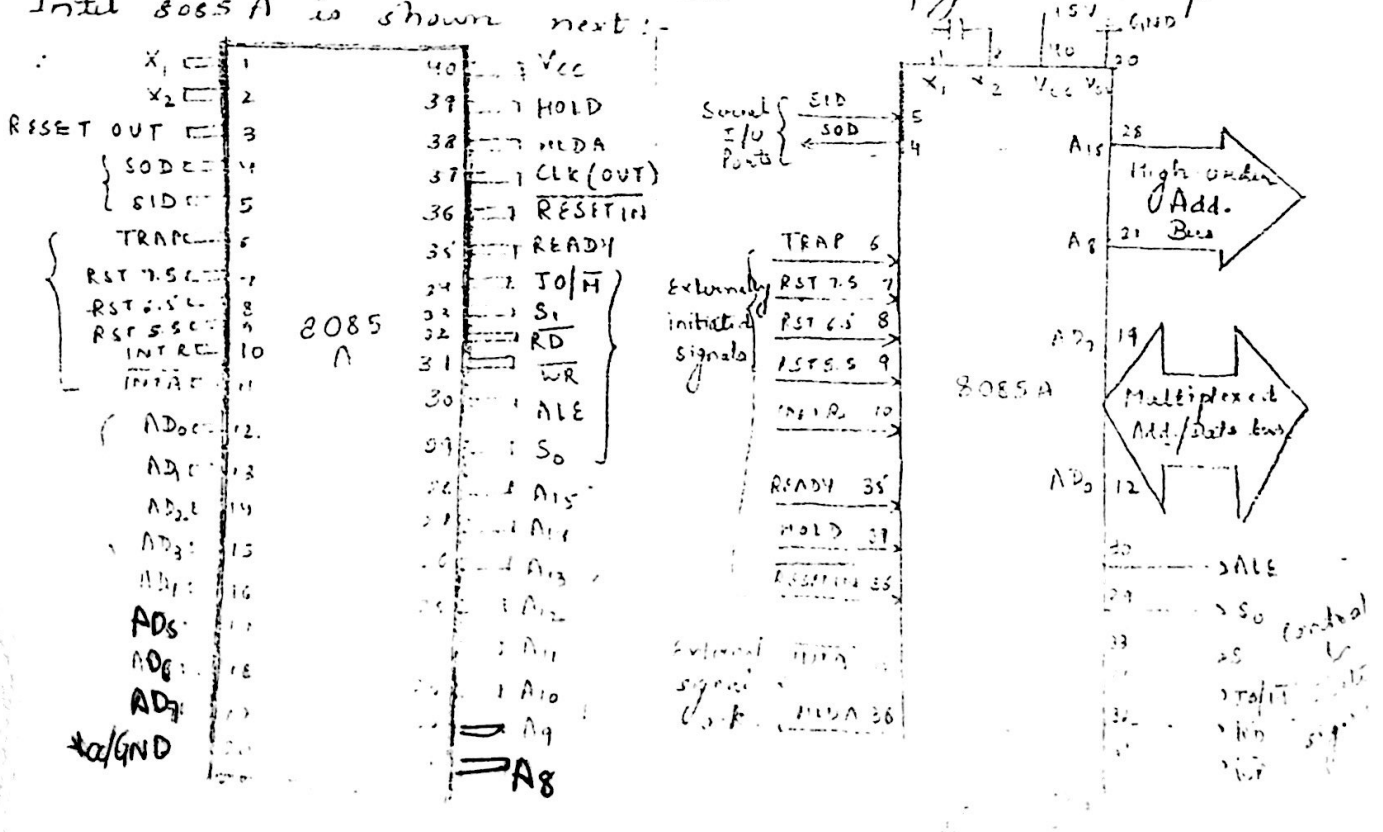
Microprocessing unit (MPU) is a device or a group of devices that can – communicate with peripherals,
- provide timing signals,
- direct data flow,
- perform computing tasks as specified by instructions in memory,
- have lines for address bus, data bus, control signals
- requires power supply & a crystal to be completely functional

8085 almost qualify as an MPU, but has two limitations as follows:-

i) Low order address bus is multiplexed (time-shared with data bus). They need to be demultiplexed.

ii) Appropriate control signals need to be generated to interface memory and I/O with 8085

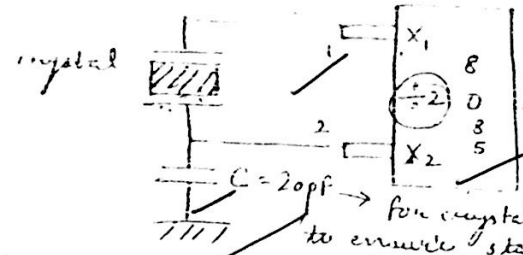
Microprocessor 8085 is a 40 pin IC. Pin configuration of Intel 8085A is shown next:-



X_1 and X_2 (Pin 1 and 2) -

These are the i/p pins of µp. A crystal (or RC, LC n/w) is connected at these two pins, which drives the internal circuitry of the µp to produce a suitable clock.

The frequency is internally divided by two, i.e. to operate a system at 3MHz (or 3.14 MHz), crystal should have frequency of 6MHz (or 6.28 MHz)



for crystals with < 4MHz to ensure starting up of crystal at the right freq.

ii) Reset out (Pin 3) - This is the o/p pin of µp. This signal indicates that MPU is being reset. The signal can be used to reset other devices.

iii) SOD (serial data out) (Pin 4) - This is the o/p pin of µp used for serial output. Data can be sent out serially by SIM instruction.

iv) SID (serial data in) (Pin 5) - This is the i/p pin of µp. This is used for i/p data serially. This is done by using RIM instruction.

v) TRAP (Pin 6) (or RST 4.5) - This is a non-maskable interrupt having highest priority.
- Its vector location is 0024H.
- This is the i/p pin.

vi) RST 7.5, RST 6.5, RST 5.5 -
- They are restart interrupts & are i/p pins of µp.
- These are the vectored interrupts that transfer the program control to specific memory locations:-

RST 5.5 location = 8 × 5.5 = 002C H
" 6.5 " = 8 × 6.5 = 0034 H
" 7.5 " = 8 × 7.5 = 003C H

They have higher priority than INTR.
Amongst them priority order; 7.5 6.5 5.5

4 5.5 × 8 = 44 0

8 × 6.5

44 | 2 = 22

This is used as a general purpose interrupt.
It has the lowest priority.
When it goes high, PC doesnot increment; it suspends its normal sequence of instruction & µp executes Interrupt Service Routine (ISR).

viii) $\overline{INTA}$ (Interrupt Acknowledge) (Pin 11) - It is the o/p pin of µp to acknowledge interrupt; an active low signal is sent on this pin.

ix) $AD_0 - AD_7$ (pin 12 to 19) - Multiplexed address/data bus -
- $AD_7 - AD_0$ are bidirectional
- They serve a dual purpose - they are used as low-order address bus as well as the data bus
- In executing an instruction, they are used as 8-bit of memory address (low-order address bus) during first clock cycle of a m/c cycle. During second & third cycle, they are used for data. This is also known as multiplexing the bus.

However, low-order bus must be latched, this is done by ALE signal.

x) V_{SS} (pin 20) - Ground Reference

xi) $A_8 - A_{15}$ (pin 21-28) - They are unidirectional & used as high order address bus.

xii) S_0 (Pin 29) & S_1 (Pin 33) -
S_0 & S_1 are status signals. They can identify various operations but they are rarely used in small systems.

xiii) IO/$\overline{M}$ (Pin 34) - It is the o/p pin of µp. It is used to differentiate b/w I/O and memory operations.
If IO/$\overline{M}$ = 1 I/O device is selected
IO/$\overline{M}$ = 0 Memory operation is performed

iv) ALE (Address Latch Enable) (Pin 30) – ALE

It is a +ve going pulse generated every time 8085 begins an operation (m/c cycle); it indicates that bits on $AD_7 - AD_0$ are address bits.

Imp This signal is primarily used to latch low-order address from multiplexed bus to generate a separate set of eight address lines, $A_7 - A_0$.

xv) $\overline{WR}$ (Write) (Pin 31) –

This is a write control signal (active low). This indicates that data on data bus are to be written into a selected memory or I/O location.

xvi) $\overline{RD}$ (Read) (Pin 32) –

This is a read control signal (active low). This indicates that selected I/O or memory device is to be read & data are available on the data bus.

8085 Machine cycle status and control signals

Machine cycle	Status $IO/\overline{M}$	S_1	S_0	Control Signals
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$
I/O write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	RD = $\overline{WR}$ = L & $\overline{INTA}$ = H
Hold	Z	X	X	
Reset	Z	X	X	

xvii) Ready (Pin 35) – This pin is used by the processor to check whether I/O device is ready to send or receive data.

If ready = High / low Peripheral is ready.

µP waits till ready goes high.

It is used by µp for synchronisation with the slow devices

xviii) RESET IN (Pin 36) -

It is the i/p pin; when active low signal is applied to this pin, µp is reset. Contents of PC = 0000H. It also disables interrupt flags are tri-stated (cannot be accessed)

xix) CLK (OUT) (Pin 37) -

This is the o/p pin & this signal can be used as the system clock for other devices.

xx) HOLD (pin 39) -

- A high on this pin suspends normal operations.
- It indicates that another device such as a DMA (Direct memory access) controller is requesting the use of address & data bus.

Bus is given to the device as soon as current m/c cycle is completed after receiving the HOLD request. Processor acquires the buses as soon as HOLD signal is removed.

xxi) HLDA (Hold acknowledge) (Pin 38) -

It is the o/p signal for HOLD ack. It indicates that HOLD request has been received.

xxii) Vcc (Pin 40) - It is +5V power supply.

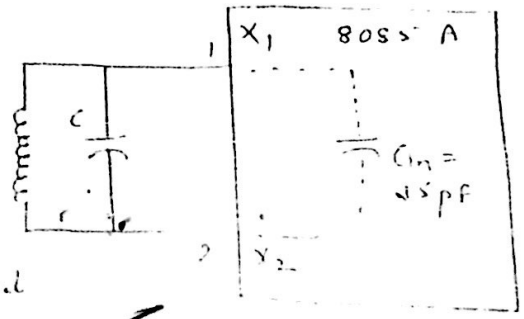
*: Clock generation methods -

1) using a parallel - resonant LC circuit -

→ $f = \frac{1}{2\pi\sqrt{L(C + C_{in})}}$

→ To minimize variation in frequency, ($C \gg 2C_{in}$ = 30pf)

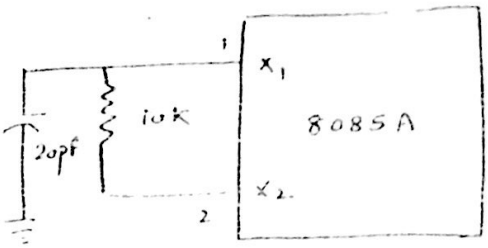
→ Use of LC ckt is not recommended for external frequency more than 5MHz.



using RC circuit –

→ This is chosen as clock source if an accurate clock freq. is of no concern.

→ Its advantage is low component cost.

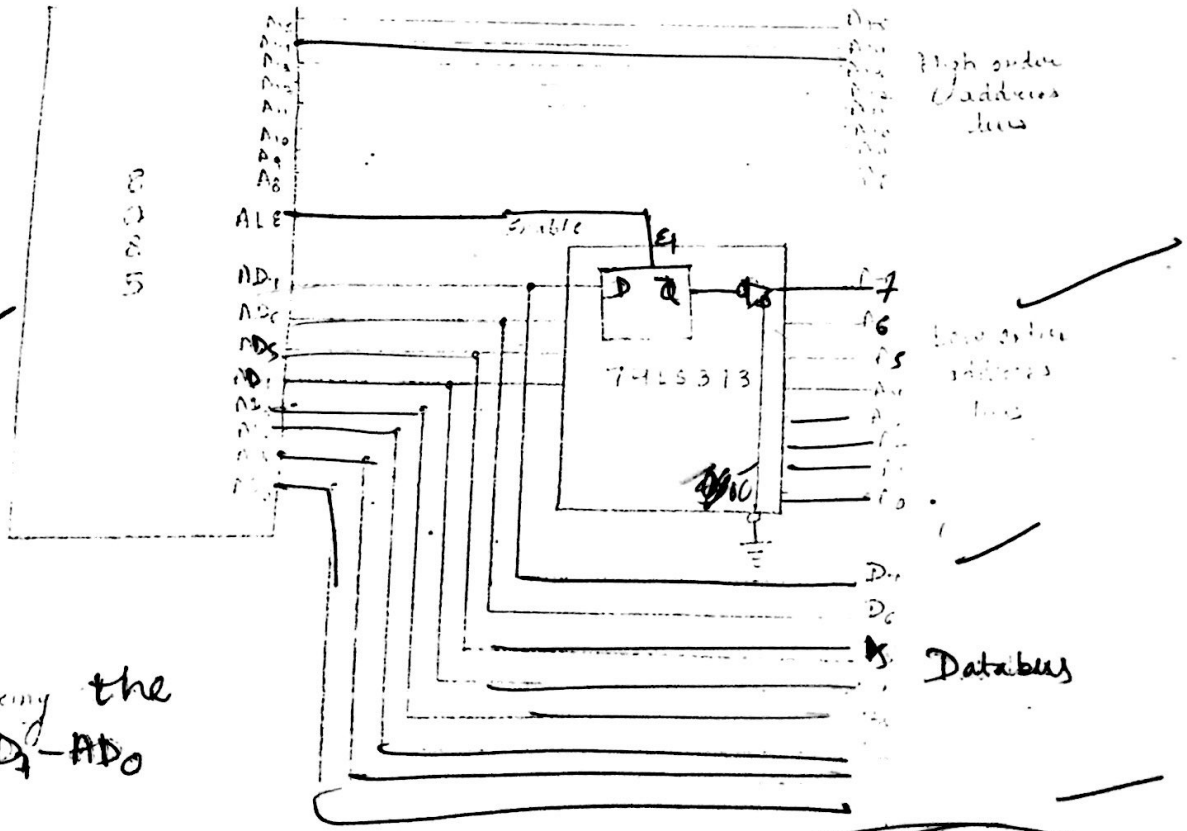


→ This ckt. is for generating 3MHz; freq. higher or lower than 3MHz should not be attempted on this.

Demultiplexing the bus AD_7-AD_0 –

- used for sending low order address as well as data.
- during first clock cycle address is transferred on these lines & during 2nd & 3rd clock cycle data is transferred on these lines
(⇒ Lower address must be latched during first clock cycle otherwise A_0-A_7 will be lost)

AD0-AD7



Phone

Demultiplexing the bus AD_7–AD_0

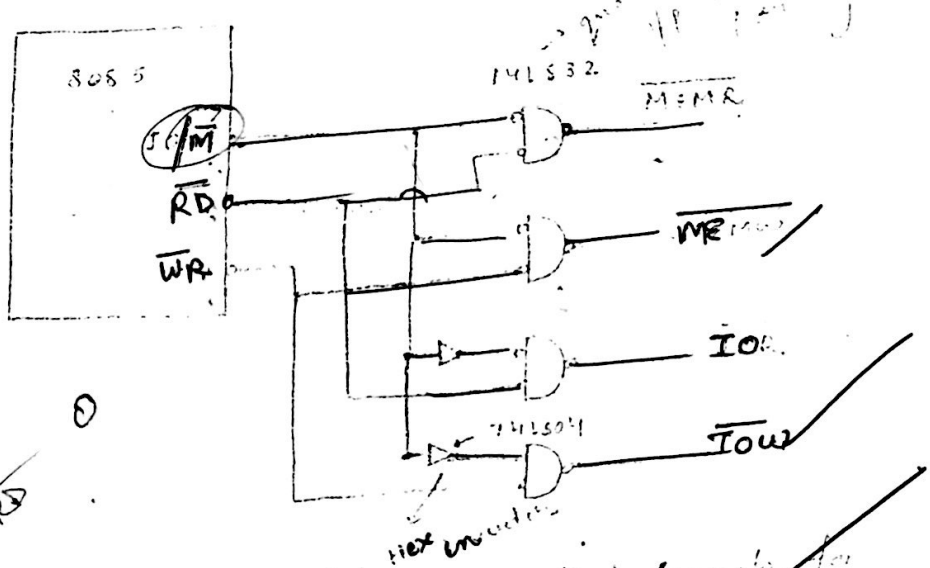
Imp → 74LS373 Latch

Generating control signals

$\overline{RD}$ (Read) is control signal that is used for both reading memory & for reading an i/p device. So, it is necessary to generate two diff. read signals :- one for memory & other for input.

Similarly, two separate write signal must be generated.

four control signals are generated by combining $\overline{RD}$ $\overline{WR}$ & IO/$\overline{M}$ as shown below.



Schematic to generate read/write control signals for memory and I/O

1. IO/$\overline{M}$ is low for memory operation
2. This signal is ANDed with $\overline{RD}$ & $\overline{WR}$ signals by using bubbled i/p NAND gates. When both i/ps are low, o/p of gate goes low to generate $\overline{MEMR}$ and $\overline{MEMW}$
3. IO/$\overline{M}$ is high indicating peripheral I/O. $\overline{IOR}$ & $\overline{IOW}$ are generated by combining IO/$\overline{M}$ with $\overline{RD}$ & $\overline{WR}$ signals as shown.

...μp processes only binary nos. but real world operates in decimal nos. alphabets & characters commonly used codes are - ASCII, BCD

struction (data format & storage) - SIGNED & UNSIGNED integers.

Instruction - An instruction is a command to the μp to perform a given task on specified data.

provide info. to the processor by memory

storage of binary bits

Each instruction has two parts

one is the task to be performed called the operation code (opcode)

second is the data to be operated on called the operand

Imp: This data can be specified in various ways. It may include 8-bit (or 16-bit) data an internal register, a memory location or an 8-bit (or 16-bit) address. In some instructions, operand is implicit.

ex:- MOV A, C

opcode operand

Types of 8085 instructions (or Instruction Word Size) -

Size

8085 instruction set is classified into the following three groups acc. to the word size or byte size

i) 1-byte instructions (or single byte) SBI

ii) 2-byte instructions (or double byte) DBI

iii) 3-byte instructions (or triple byte) TBI

i) one-byte instructions - A 1-byte instruction includes the opcode & the operand in the same byte. Here, no. is not given as operand in the instruction.

A mnemonic followed by a letter (or 2)

instruction can be ... by exposed ... byte of ... code

	opcode	operand	Binary code	hex code
ex:-	MOV	C, A	0100 1111	4F H
	ADD	B	1000 0000	80 H
Invert contents of acc.	CMA		0010 1111	2F H

ii) 2-byte instruction -

In a 2 byte instruction, first byte specified the operation code & second byte specifies the operand. Here, a 8-bit no. is there in the instruction. "A mnemonic followed by 8-bit"

To specify an instruction completely 2 bytes are required (as 2 8 bit codes)

ex:-	MVI	A, 32H	0011 1110	3E H
			0011 0010	32 H
Compare immediate with accumulator	CPI	24 H		

iii) 3-byte instruction -

In a 3-byte instruction, first byte specifies the opcode & following 2 bytes specify the 16-bit address. Second byte is low-order address & 3 byte is high-order address.

Here, 16-bit no. is there in the instruction

"A mnemonic -followed by 16-bit."

ex:-	LXI	H, 1536H	0010 0001	21 H (first byte)
			0011 0110	36 H (second byte)
			0001 0101	15 H (third byte)
Load register pair immediate				
	LDA	2050H	0011 1010	3AH
			0101 0000	50 H
			0010 0000	20 H
Load in accumulator				

Opcode format - (how an instruction is designed into the µp)

All internal registers are identified as follows:-

Code	Registers
000	B
001	C
010	D
011	E
100	H
101	L
111	A
110	Reserved for memory related op

Code	Reg. pairs
00	BC
01	DE
10	HL
11	AF or SP

ADD B

ex: ADD B
10000 000 = 80H

ex: MOV A, C
01 111 001 = 79H

MOV C, A
01 001 111 = 4FH

MOV Rd, Rs

0	1	D	D	D	S	S	S

Imp:

Addressing modes of 8085 -

To perform any operation using µp, we have to give corresponding instruction to the µp for executing any instruction µp requires data.

The various formats of specifying the operands are called A. modes"

The methods by which address of source of data is given in an instruction is called as addressing mode

There are 5 types of addressing modes of 8085 -

(1.) Immediate addressing mode - (Pass the button)

If 8/16 bit data is given alongwith the instruction In such instructions last alphabet of mnemonic will be generally 'I'.

ex: MVI A, 35H 35H → A

LXI B$, 4000H

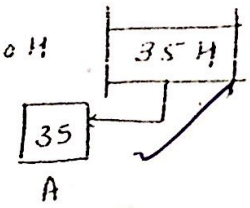
L MVI A, 30H
MVI

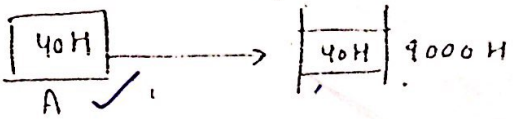
(2) Direct addressing mode - (combination no.17 on the menu)

If 8/16 bit data is present in memory location & this 16 bit address of this location is given alongwith the instruction.

ex:- LDA 3000H

STA 9000H





(3.) Register direct addressing mode - (pass the baton)

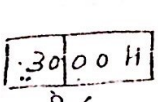
If 8/16 bit data is present in register / register pair name of reg. / reg. pair is given alongwith the instruction.

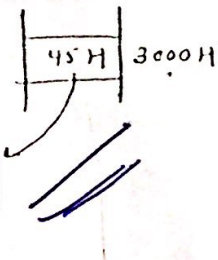
ex:- MOV A, B

(4.) Register indirect addressing mode - (I will what Susie has)

If 8/16 bit data req. for executing instruction is present in memory location & this 16-bit address is present in reg. pair & this reg. pair is given alongwith the instruction

ex:- LDAX B





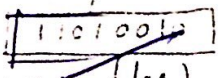
STAX B

(5.) Implicit addressing mode - (Let us eat)

If add. of source of data as well as destination is fixed, then there is no need to give operand alongwith the instruction ex -

CMA

STA A



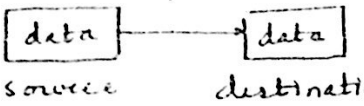
Instruction set classification

μp can perform maximum 256 different operations. The diff. instructions of μp are divided into following types:-

1) Data transfer (copy) instructions
2) Arithmetic instructions
3) Logical instructions
4) Branching instructions
5) Machine control instructions

Data transfer (copy) operations -

This group of instructions copy data from a location called source to another location called destination, without modifying the contents of source.



Actually, these instructions are not data transfer but data copy instructions b'coz source is not modified.

Available data transfer -

Data available (source)		Data to transfer (destination)	
Im		R, M	
R		R, M	(M → M is not possible)
M		R	
I O		A	Always with acc.
A		IO	

* In any instruction, if M is present, then the address of memory location will be always given by HL pair.

This group contains 16 instructions :-

(1) MOV : Move — Copy from Source to destination

MOV Rd, Rs — MOV A, B

MOV M, Rs — MOV M, A

MOV Rd, M — MOV B, M

) MVI : Move Immediate 8-bit

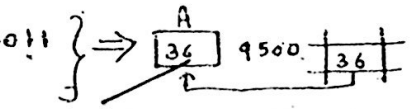
MVI Rd, Data — MVI B, 28H

MVI M, Data — MVI M, 20H

(3.) LXI : Load Register Pair Immediate

LXI Reg. pair, 16-bit data — LXI B, 2050H

ex: LXI H, 9500H
MOV A, M

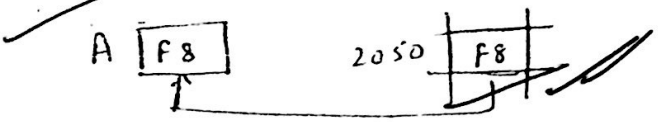


{B = 20H, C = 50H}

(4) LDA : Load accumulator direct

LDA 16-bit address — LDA 2050H — 3A 50 20

Rev. order in num.

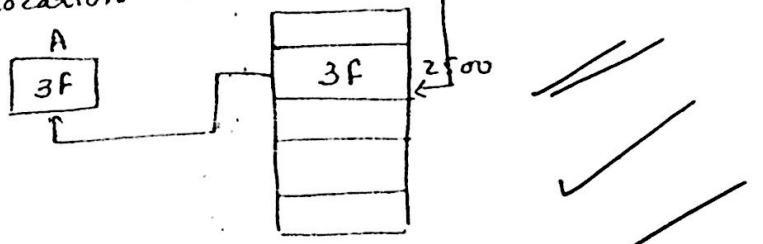


(5.) LDAX : Load Accumulator Indirect

LDAX Reg. pair — LDAX B

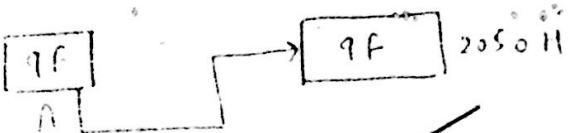
Reg. pair points to a memory location

(B/D)



(6) STA : Store accumulator direct

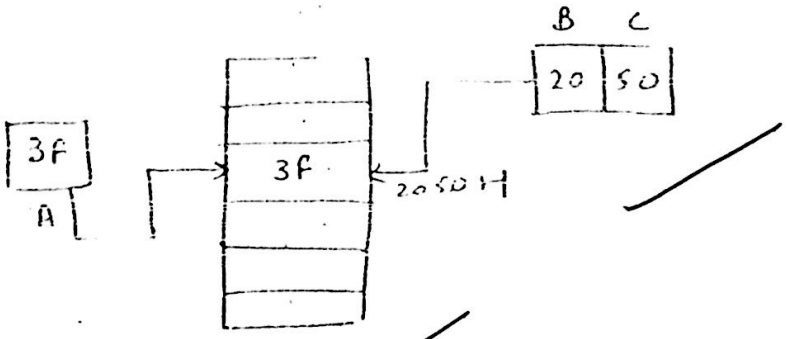
STA 16-bit add. — STA 2050 H



(7.) STAX : Store accumulator indirect

STAX B/D reg. pair

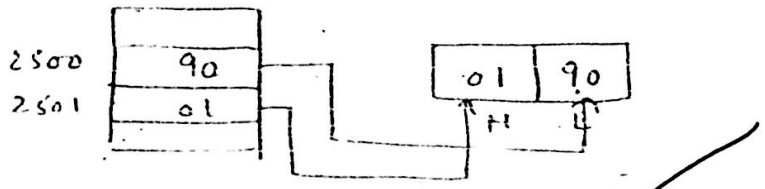
STAX B



(8.) LHLD : Load H and L Registers Direct

LHLD 16-bit address

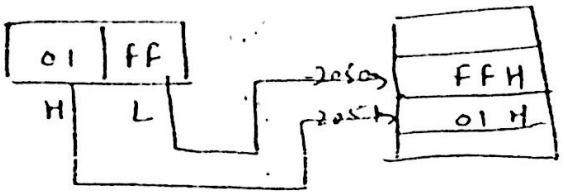
LHLD 2500H



(9.) SHLD : Store H and L Registers Direct

SHLD 16-bit address

SHLD 2050H



(10.) XCHG: Exchange H and L with D and E -

XCHG (No operand)

H ⟷ D
L ⟷ E
} This transfer uses temporary registers.

W
Z

HL ⟶ WZ
DE ⟶ HL
DE ⟵ WZ

(11.) SPHL : Copy H and L Registers to the Stack Pointer -

SPHL (None)

H provides high order address
L " low order address

(12) PCHL : Load Program counter with HL contents

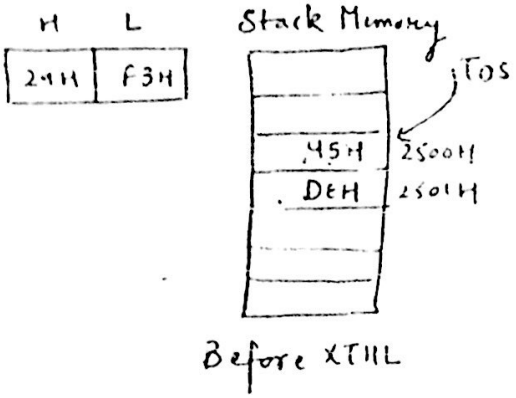
PCHL (None)

H provides high-order byte
L provides low-order byte

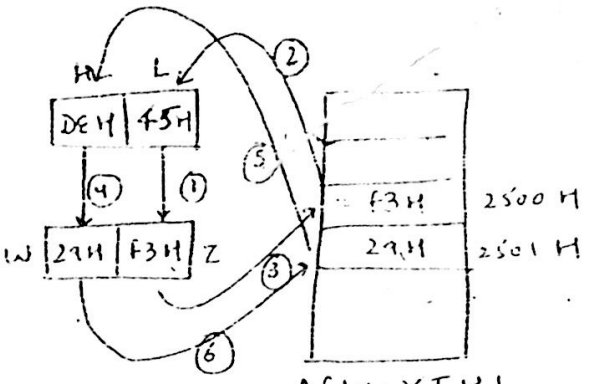
(13) XTHL : Exchange H and L with Top of stack

XTHL (None)

L ⟷ Contents of SP register
H ⟷ " " SP+1



Before XTHL



After XTHL

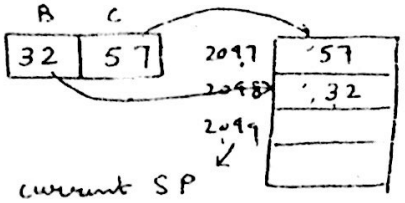
Execution of XTHL

(14) PUSH : Push register pair onto stack or PUSH PSW

PUSH Reg. pair

PUSH PSW (AF)

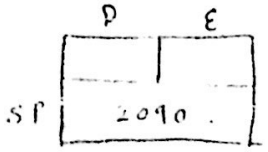
PUSH B



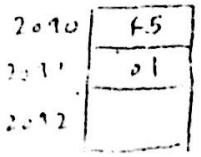
i) Decrement SP - 2098
ii) copy contents of high order register (B, D, H or A)
iii) Decrement SP = 2097
iv) copy contents of low order register (C, E, L or F)
v) New value of SP = 2096

(15) POP Reg. pair / PSW - Pop off stack to Reg. pair

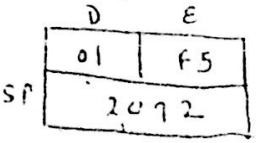
POP D



Stack contents



Reg. content after instruction



1) OUT: Output data from Accumulator to a port with 8-bit address -

OUT 8-bit port address

OUT 0FH

2) IN: Input data to accumulator from a port with 8-bit address

IN 8-bit port address

IN F8H

Arithmetic Instructions -

8085 has 14 arithmetic instructions. It can perform 8-bit binary & decimal addn. It can perform 16-bit binary addn. 8-bit subtraction can be performed.

But it cannot perform 16-bit BCD addn & BCD (8-bit) subtn by single instruction

It can perform 8-bit & 16-bit inc./dec. operations

(1) ADD: Add register to Accumulator

ADD Reg.
M

ADD B.
ADD M (HL contains the address)

Program 1:- Write a program to add the contents of memory locations 9510 and 9511. Neglect carry. Store the result in memory location 9512.

LDA 9510H
MOV B, A
LDA 9511H
ADD B
STA 9512H

LDA=

(2) ADC: Add register to accumulator with carry

ADC Reg.
M

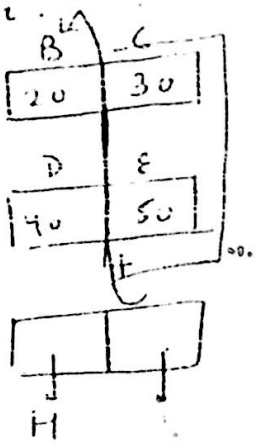
ADC L
ADC M

Ques 2. WAP to add two 16-bit nos: 2030H and 4050H using ADD & ADC instructions, neglecting the carry. Store the result in H and L register.

(STC, CMC ; to clear carry)

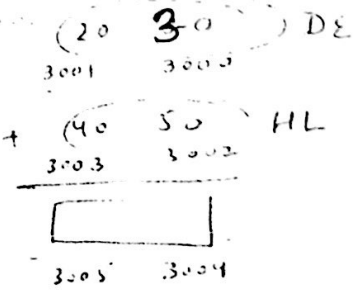
Immediate addressing mode

```
LXI B, 2030H
LXI D, 4050H
MOV A, E
ADD C
MOV L, A
MOV A, D
ADC B
MOV H, A
```

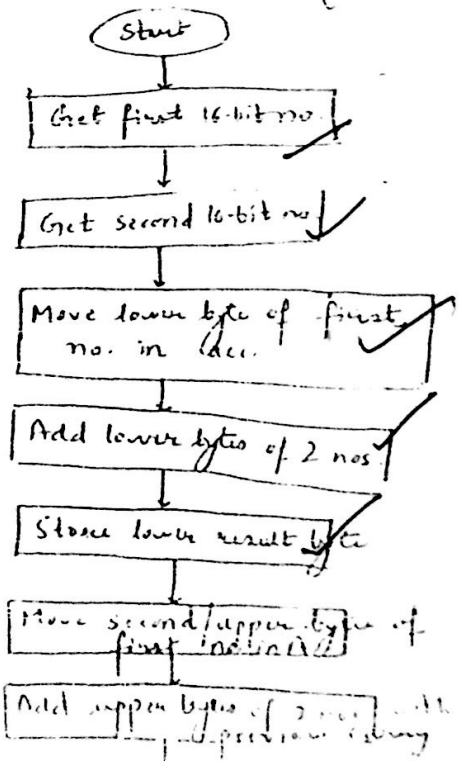


Ques 1. Try this program with direct addressing mode.

```
LHLD 3000H
XCHG ;  HL → DE
LHLD 3002H
MOV A, E
ADD L
MOV L, A
MOV A, D
ADD H
MOV H, A
SHLD 3004H
```



Flow chart (Immediate)



ADI: Add immediate to accumulator -

ADI 8-bit data ADI 59H (A ← A+59)

: Add immediate to accumulator with carry

ACI 8-bit data ACI 57H (A = A+57+CY)

Imp (5) DAD : Add Reg. pair to H and L Registers

DAD Reg. pair

The 16-bit contents of specified reg. pair are added to the contents of the HL register & the sum is stored in HL reg. If result is larger than 16 bits, CY flag is set. No other flags are effected.

ex:- Assume HL pair contains 0242H. Multiply its content by 2

We can have, DAD H

ex- DAD SP

Program 3:- To add two 16-bit nos. & store the result in HL pair

LXI H, F378H

LXI D, 45D0H

DAD D

(Immediate addressing mode)

DAD

Imp (6) DAA : Decimal Adjust Accumulator

DAA (None)

It changes the contents of acc. from a binary value to two 4-bits BCD. This is the only instruction that uses AC to perform binary to BCD conversion.

How?

A = D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 → 1001

If > 9 (1001 + 6) If > 9 or AC = 1 (+6)

DAA converts binary sum of two BCD nos in BCD but it cannot convert binary sum of two binary nos to BCD.

The instruction is only to be used after 8-bit add operation

1 ← 2 ← 4

0 1 0 0 0 1 0 1

Program 4 :- WAP to add two BCD nos. 45 & 95.

```
MVI A,45
ADI 95
DAA
```

45 → 6 → 0 → 3 → 1

$1\times2^0 + 0\times2^1 + 1\times2^2 + 1$

45	0 100 0101
95	1 001 0101
	1 101 1010 CAH
	+ 0 110 0110
CY	1 0011 0000 40H

(7.) SUB : Subtract reg. or memory from accumulator

SUB Reg. / Mem. — SUB C — (A = A - C (using 2's complement))

(8.) SBB : Subtract source & borrow from accumulator

SBB Reg. / Mem. — SBB D — (A = A - D - C)

(9.) SUI : Subtract immediate from accumulator

SUI 8-bit data — SUI 37H — (A = A - 37H)

(10.) SBI : Subtract immediate with borrow

SBI 8-bit data — SBI 25H — (A = A - 25H - CY)

45

(11.) INR : Increament contents of Register / Memory by 1

INR Reg. — INR D — (D = D + 1)

Mem. — INR M — (M = M + 1, HL remains same)

(12.) INX : Increment Register Pair by 1

INX Reg. pair — INX H

10110 1

22
45
2

(13.) DCR : Decrement source by 1

DCR Reg. / Mem. — DCR B

(14.) DCX : Decrement Reg. pair by 1

DCX Reg. pair — DCX D

2	39	19	9	4	2	1	0	1
2	89	44	22	11	5	2	1	0
2	44	22	11	5	2	1	0	1
2	22	11	5	2	1	0	1	0
2	11	5	2	1	0	1	0	1
2	5	2	1	0	1	0	1	0
2	2	1	0	1	0	1	0	1
	1	0	1	0	1	0	1	0

Program 5: WAP to subtract 16 bit data stored at memory location 2003 : 2002 H from another 16-bit data stored at 2001 : 2000 H. Neglect borrow, if any. Save the result in memory location 2005 : 2004 H.

LHLD 2000H : Load HL with data from 2001 : 2000

XCHG : Data from HL is transferred to DE

LHLD 2002H : HL = no. from 2003 : 2002

MOV A, L

SUB E

MOV L, A
MOV A, H
SBB D
MOV H, A
SHLD 2004H

or

STA 2004 H
MOV A, H
SBB D
STA 2005 H

We can also use

LXI H, 2000H
MOV E, M
INX H

Logical Instructions

8085 can perform 6 logical operations. These are NOT, OR, AND, EXOR, Compare & rotate operations. Except compare, all the other five operations are performed on bit by bit basis & hence these five operations are also called as bit manipulation operations.

① CMP : Compare with accumulator

CMP Reg.
Mem.

If (A) < (Reg/Mem): CY = 1 and Z = 0

If (A) = (Reg./Mem.): Z = 1 and CY = 0

If (A) > (Reg./Mem.): CY = 0 and Z = 0

(2) CPI : Compare immediate with Accumulator

CPI 8-bit data

CPI 98H

If A < Data : CY=1 and Z=0

If A = Data : Z=1 and CY=0

If A > Data : CY=0 and Z=0

(3) ANA : Logical AND with Accumulator

ANA Reg.
Mem.

ANA D

A 54 → 0 1 0 1 0 1 0 0

D 82 → 1 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0

D7 D6 D

⇒ A = 00
⇒ D = 82

S=0 ✓
Z=1 ✓
P=1 ✓
AC=1 –
CY=0 –

(4) ANI : AND immediate with accumulator

ANI 8-bit data

ANI 97 H

A = A3H → 1 0 1 0 0 0 1 1

1 0 0 1 0 1 1 1

1 0 0 0 0 0 1 1

A = 83

F = 1 0 0 1 0 0 0 0
S Z AC P CY

(5) XRA : Exclusive OR with accumulator

same = 0
diff = 1

XRA reg.
mem.

XRA D

77H → 0 1 1 1 0 1 1 1

56H → 0 1 0 1 0 1 1 0

0 0 1 0 0 0 0 1

A = 21 H

S=0, Z=0, P=1,
CY=0, AC=0

(6) XRI : Exclusive OR Immediate with accumulator

XRI 8-bit data

XRI A211

CMP

(7.) ORA : Logically OR with accumulator

ORA Reg. / Mem.

ORA C

A	03H	0000 0011
C	81H	1000 0001
		1000 0011

A = 83H
S = 1, Z = 0, P = 0,
CY = 0, AC = 0

(8.) ORI : Logically OR Immediate

ORI 8-bit data

ORI 86H

(9.) CMA : Complement Accumulator

CMA (none)

A = 89H 1000 1001

CMA

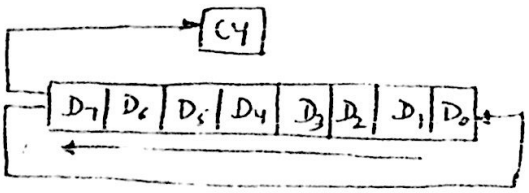
A = 76H 0111 0110

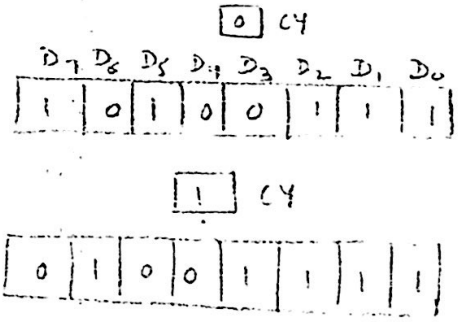
(10.) RLC : Rotate Accumulator Left

RLC (none)

Each binary bit of the accumulator is rotated left by one position. Bit D_7 is placed in the position of D_0 as well as in the carry flag.

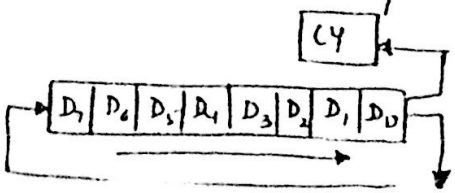
CY is modified but S, Z, P, AC are not affected.



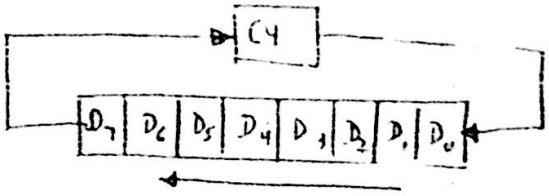


(11.) RRC : Rotate accumulator right

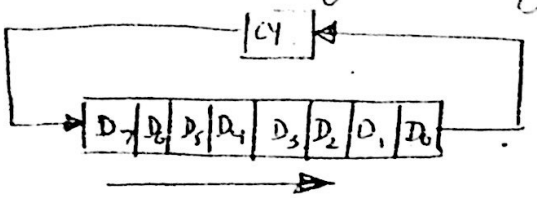
Each binary bit of accumulator is rotated right by one position. Bit D_0 is placed in D_7 as well as in CY flag.



(12) RAL : Rotate accumulator left through carry -



(13) RAR : Rotate accumulator right through carry -



(14) CMC : Complement carry

CMC (None)

CY flag is complemented. No other flag is affected.

(15) STC : Set Carry

STC (None)

CY = 1. No other flags are affected.

Program 6 : A packed BCD number is stored in memory location 2000H. Unpack the BCD no. and store the two digits in memory locations 2001H and 2002H such that memory location 2001H will have lower BCD digit.

```
LXI H, 2000H : (HL is initialized as a memory pointer)
MOV A, M     : Get packed BCD no.
MOV B, A     : Move no. in B reg.
ANI F0H      : Mask lower nibble
RRC
RRC
RRC
RRC          Adjust higher BCD as lower nibble
INX H
```

```
INX H          : HL = 2002 H
MOV M, A
MOV A, B
ANI 0FH
DCX H          : HL = 2001 H
MOV M, A       : Store lower BCD at 2001 H
```

Ques.3 Pack the two unpacked BCD nos. stored in memory locations 2000H and 2001H and store result in memory location 2003H. Assume most significant digit is stored at 2000H and least significant digit is stored at 2001 H.

```
LXI H, 2000H
MOV M, H
RLC
RLC
RLC
RLC
ANI F0H  : Make least significant BCD 0
MOV C, A
INX H
MOV A, M
ADD C
INX H
MOV M, A
```

Branching Instructions -

These instructions are also called as control tranfer instructions as the control of operation is transferred from one location to another location.

There are 3 categories of instructions by which control is transferred to -

i) another location within the same program (JUMP)

ii) another program called sub-routine (CALL)
iii) from sub routine program to main program (RET).

This control may be :-

- unconditional
- conditional

It uses sign, zero, parity & carry flag bits. (Not AC).
On the basis of setting & resetting of these flag bits, we can have 8 diff. conditions.

① JMP : Jump unconditionally

JMP 16-bit add. — JMP 2034H

② Jump conditionally

				M-cycles / T-states
JC	16-bit address	Jump on carry	CY=1	2M/7T (if condition is not true)
JNC	"	Jump on no carry	CY=0	
JP	"	Jump on positive	S=0	3M/10T (if condition is true)
JM	"	Jump on minus	S=1	
JPE	"	Jump on parity even	P=1	
JPO	"	Jump on parity odd	P=0	
JZ	"	Jump on zero	Z=1	sequence is continued in time requires fewer T-states
JNZ	"	Jump on no zero	Z=0	

③ CALL : Unconditional subroutine call

CALL 16-bit address

Program sequence is transferred to the memory location specified by the 16-bit address given in the operand.

Before the transfer, address of next instruction after CALL is pushed onto the stack.

Machine control Instructions -

① NOP : No operation

NOP (None)

This instruction is fetched & decoded ; however, no operation is executed.

This is used to fill in time delays or to delete/insert instructions while troubleshooting.

② HLT : Halt and enter wait state

HLT (none)

MPU finished executing the current instruction & halts any further execution.

MPU enters halt acknowledge m/c cycle & wait states are inserted in every clock period. Address & data bus are placed in the high impedance state.

Contents of registers are unaffected during the HLT state.

An interrupt or reset is necessary to exit from the Halt state.

③ DI : Disable Interrupts

DI (None)

Interrupt enable ff is reset & all the interrupts except TRAP (8085) are disabled.

This is used when users donot want that system must not be interrupted.

④ EI : Enable Interrupts

EI (None)

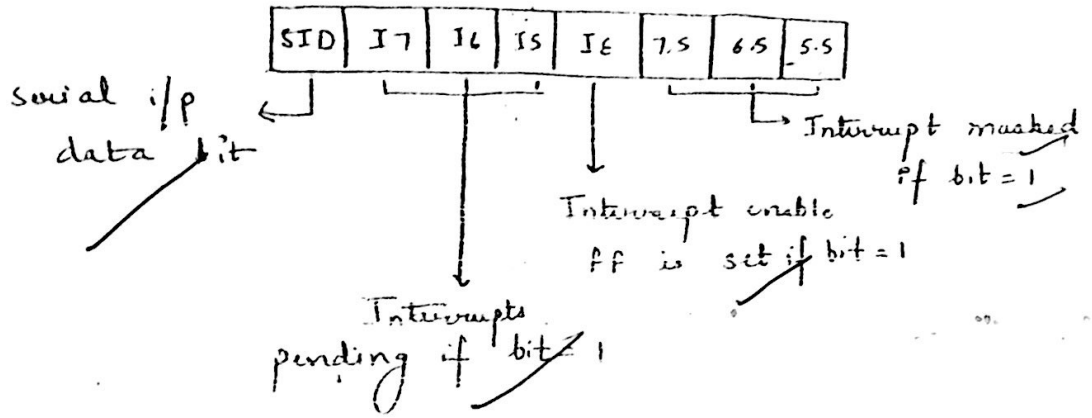
The interrupt enable ff is set & all interrupts are enabled.

After a system reset, EI is reset, thus disabling the interrupts. This instruction is necessary to re-enable the interrupts (except TRAP).

RIM : Read Interrupt Mask

RIM (None)

This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5 and 5.5 & to read serial data i/p bit. This instruction loads eight bits in the A :-



ex:- After executing RIM, A = 49 H

0 1 0 0 1 0 0 1

RST 7.5 is pending

Interrupt enable ff = 1

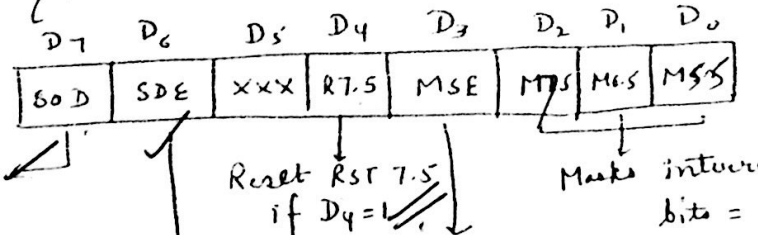
RST 5.5 is masked

SID I7

SIM : set Interrupt Mask-

SIM (None)

This is a multipurpose interrupt & used to implement software interrupts (RST 7.5, 6.5 and 5.5) and serial data output



serial o/p data

made available to line if $D_5 = 1$

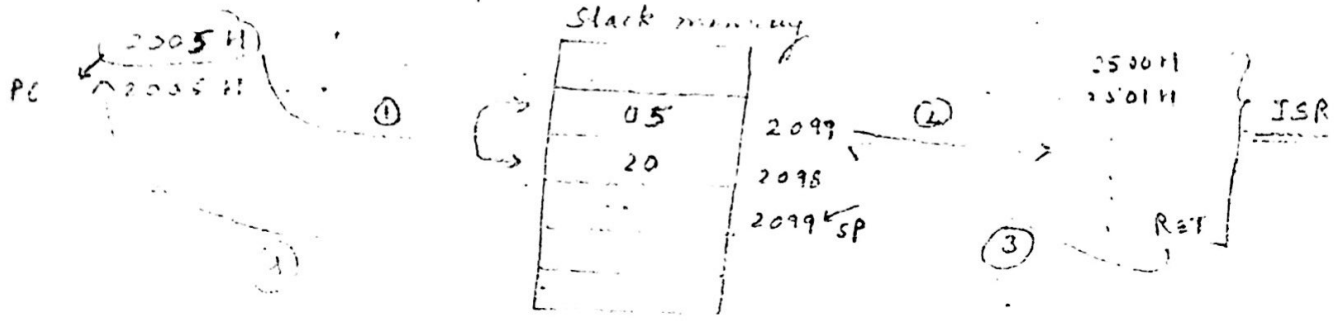
Serial data enable
1 = Enable
0 = Disable

Reset RST 7.5 if $D_4 = 1$

Mask set enable if D3 = 1.
(If D3 = 1, functions of D2, D1 & D0 are enabled)
This is a master control over all interrupt masking bits.
If D3 = 0, D2, D1, D0 don't have significance

Mask interrupts if bits = 1

Sequence of operations in CALL



(4.) Call conditionally – (operand : 16 bit add.)

CC	Call on carry	CY = 1
CNC	Call with no carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0
CZ	Call on zero	Z = 1
CNZ	call on no zero	Z = 0

M-cycles / T states

2/9 if cond. is not true

5/18 if cond. is true

(5.) RET : Return from subroutine unconditionally

RET (None)

2095 50

2096 20

After executing RET,

PC = 2050H

SP = 2097H

(6.) Return conditionally (No operand)

RC	Return on carry	CY = 1
RNC	Return with no carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0

M-cycles / T-states

1/06 (cond. is not true)

3/12 (cond. is true)

(7) PCHL : Load PC with HL contents (Discussed earlier)

(8) RST 0-7 : Restart

RST instructions are equivalent to 1-byte call instructions to one of the eight memory locations on page 0.

These instructions are generally used in conjunction with interrupts & inserted using external h/w. However, they can be used as s/w instructions in a program to transfer program execution to one of the eight locations.

		Hex code	Restart Address (H)
RST 0	11 000 111	C7	0000
RST 1	11 001 111	CF	0008
RST 2	11 010 111	D7	0010
RST 3	11 011 111	DF	0018
RST 4	11 100 111	E7	0020
RST 5	11 101 111	EF	0028
RST 6	11 110 111	F7	0030
RST 7	11 111 111	FF	0038

Additional 8085 Interrupts -

The 8085 has four additional interrupts and these interrupts generate RST instructions internally & thus do not require any external hardware.

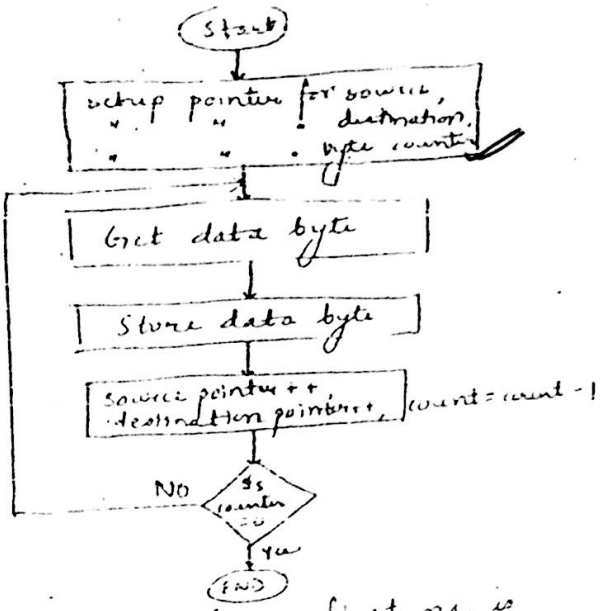
These instructions & their restart addresses are as follows -

	Restart address	
RST 4.5 / TRAP	24H	(4.5 X 8)
RST 5.5	2CH	
RST 6.5	34H	
RST 7.5	3CH	

Programs related to loops-

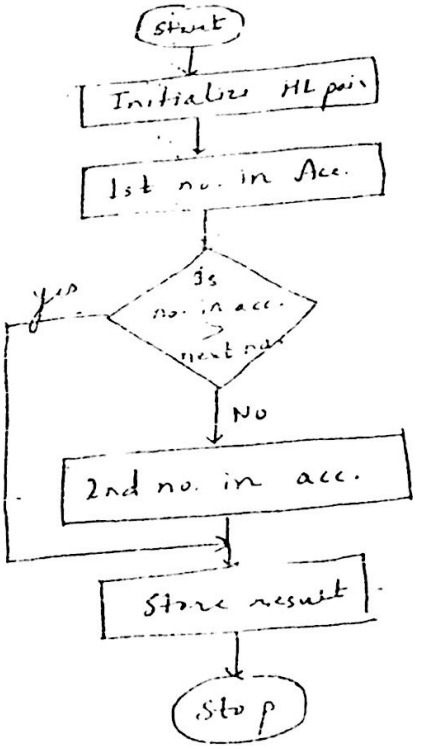
Ans. 1. WAP to transfer block of (5 bytes) starting from memory location 2000H. to starting destination address 3000H.

```
      LXI H, 2000H
      LXI B, 3000H
      MVI D, 05H
Li:   MOV A, M
      STAX B
      INX H
      INX B
      DCR D
      JNZ L1
      HLT
```



Ans. 2. WAP to find the larger of two numbers. first no. is placed in memory location 2000H & second no. is placed in memory location 2001H. Store result in memory location 2002 H.

```
      LXI H, 2000H
      MOV A, M
      INX H
      CMP M
      JNC L1
      MOV A, M
L1:   STA 2002H
      HLT
```



LXI B H 200H.

The 8085 Machine Cycles and Bus Timings

1) The 8085 is designed to execute 74 different instruction types.

2) Each instruction has two parts : opcode and operand.

- opcode → It is a command such as Add
- operand → It is the object to be operated on; such as a byte or contents of a reg.

3) Some instructions are 1-byte & some are multibyte.

4) To execute an instruction, 8085 needs to perform various operations such as Memory read/write and IO read/write.

(There is no direct relationship b/w no of bytes of an instruction & no. of operations 8085 has to perform.

5) All instructions are divided into a few basic machine cycles & these m/c cycles (M) are divided into precise system clock periods (T).

m/c cycles

Instruction = $M_1 + M_2 + \cdots + M_n$

$T_1 + T_2 + \cdots + T_m$

$T_1 + T_2 + \cdots + T_m$

clock periods.

6) Basically, µp external communication functions can be divided into three categories :-

i) Memory read / write
ii) I/O read and write
iii) Request Acknowledge

These functions are further divided into various operations (machine cycles) such as - Opcode fetch, Mem. Read, Mem Write, I/O Read, I/O Write, Interrupt ack, Halt, Hold, Reset.

7) Each instruction consists of one or more m/c cycles for

ode fetch Machine Cycle :-

first operation in any instruction is Opcode fetch (reading instruction from memory). µp fetch/gets this code in memory where it is stored before µp can begin execute the instruction.

This process can be easily understood using an analogy of how a packet is picked from house by a shipping Co. :-

i) Courier gets address from the office & finds the street & looks for your house number.

ii) Courier rings the bell.

iii) Somebody opens the door & gives package to the courier.

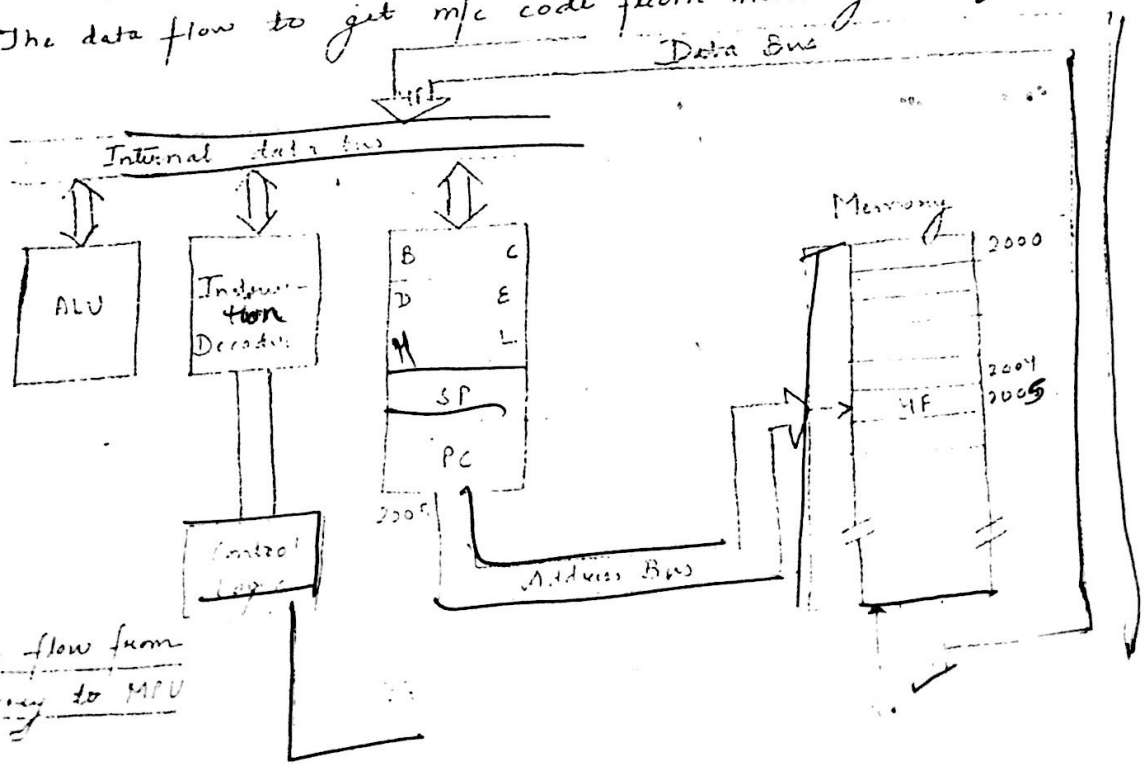
iv) Courier returns to the office with the package.

v) Office staff disposes package acc. to instructions given by the customer.

Consider the following instruction :-

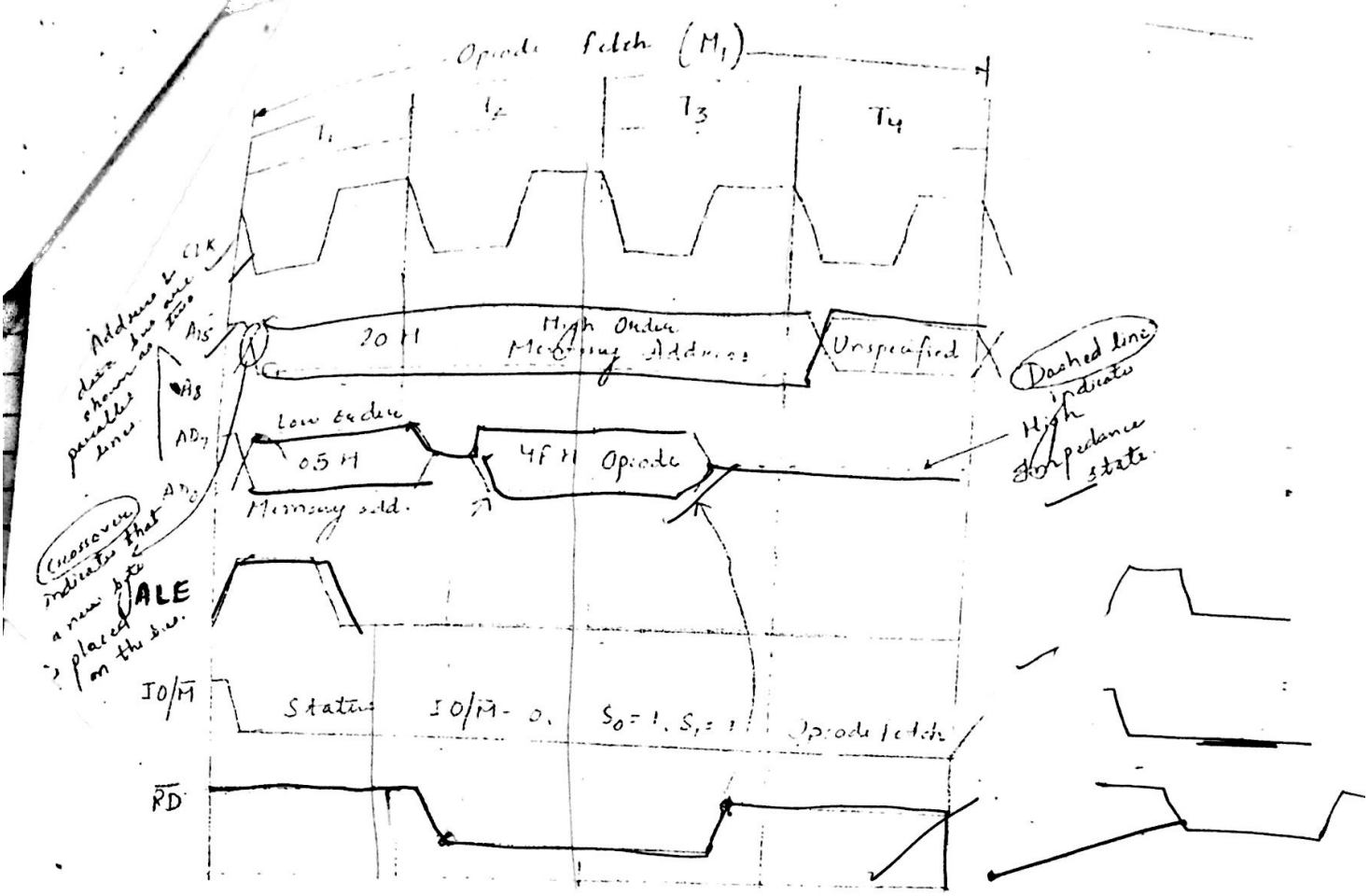
2005 : MOV C, A	0100 1111 (4FH)

The data flow to get m/c code from memory is shown :-



Data flow from memory to MPU

MOV C,A :



Timing : Transfer of byte from Memory to MPU

The above timing diagram shows how a data byte is transferred from memory to the MPU in relation to the system clock. MPU performs the following steps :-

step 1 :- The µp places 16-bit memory address from PC on the address bus (courier getting on road to find the address).

(At T_1 - 20 H is placed on A_{15}-A_8 & 05 H (low-order memory address) is placed on AD_7-AD_0 & ALE goes high Also, IO/$\overline{M}$ goes low to indicate memory related operation)

step 2 :- CU sends control signal $\overline{RD}$ to enable memory chip (Ringing the doorbell).

(At T_2 - $\overline{RD}$ is sent out. $\overline{RD}$ is active during two clock periods)

step 3 :- The byte from the memory location is placed on the data bus

(At T_2 :- 4FH is placed on $AD_7 - AD_0$ due to $\overline{RD}$. When $\overline{RD}$ goes high, bus go into the high impedance state).

Step4 :- The byte is placed in the instruction decoder of the µp & the task is carried out according to the instruction.

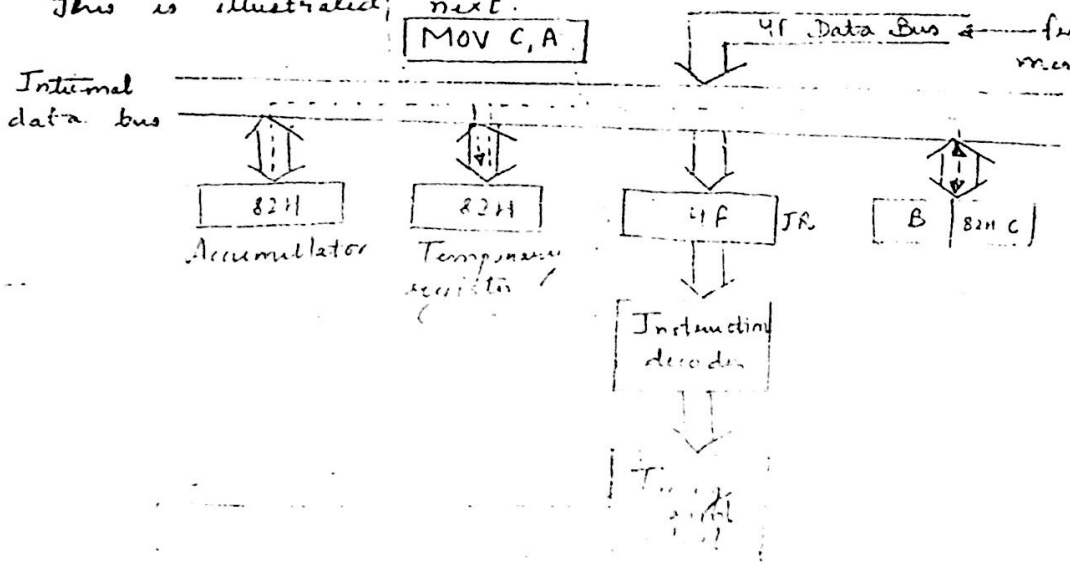
(At T_4 :- 4FH is decoded by instruction decoder & contents of A are copied into register C)

To differentiate an opcode from a data byte or an address, this m/c cycle is identified as Opcode fetch cycle ($IO/\overline{M} = 0$, $S_0 = 1$ & $S_1 = 1$).
mem. op. op fetch

This opcode fetch cycle is called M_1 cycle & has four T-states. States $T_1 - T_3$ to fetch the code & T_4 to decode & execute the opcode (Some instructions have opcodes with 6 T-states).

2. Decoding and executing an Instruction :-

After an instruction has been fetched, it is decoded & executed. This is illustrated next.



Instruction decoding and execution

i) Content of data bus (4FH) is placed in the instruction register (IR) & decoded.

ii) Transfer the contents of the A (82H) to the temporary register in the ALU.

iii) Transfer contents of the temporary register to register C.

3. Memory Read Machine Cycle -

To illustrate this, we need to examine execution of a 2-byte/3-byte instruction b'coz in a 1-byte instruction m/c code is an opcode, ∴ operation is always an opcode fetch.

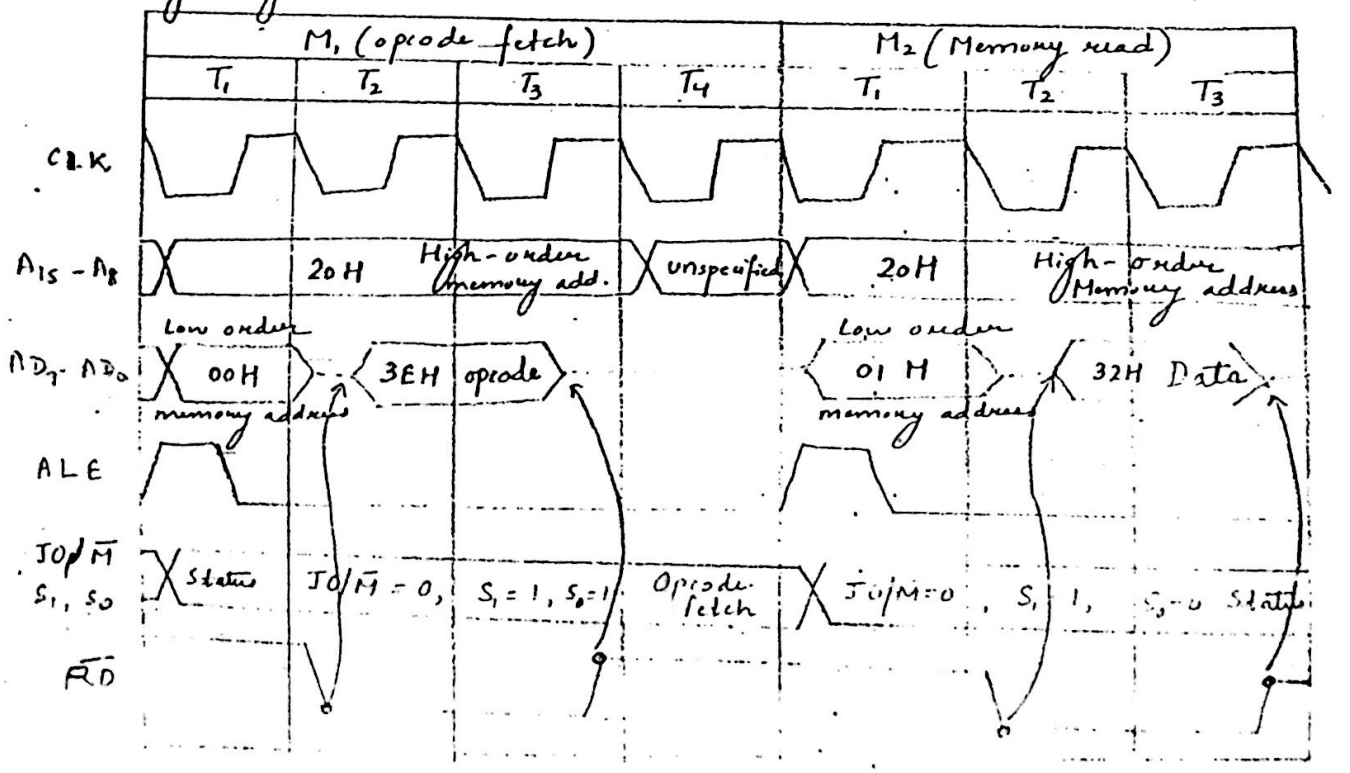
Consider | 2000H : MVI A, 32H | ; Load 32H in the acc.

⇒ 2000H : 3EH (MVI A) ← opcode
2001H : 32H ← data byte

⇒ To execute this instruction, we require two m/c cycles.

| M_1 = opcode fetch | (4 T-states) | } Total T-states = 7
| M_2 = memory read | (3 T-states) |

Timing diagram is shown next:-



Step 1:- First m/c cycle (M_1) (opcode fetch) is executed.

At T_1:- 011 ($IO/\overline{M}, S_1, S_0$) is placed on status signals
20H on A_{15} - A_8
00H on AD_7 - AD_0
PC = PC+1 (2001H)
ALE goes high.

At T_2:- $\overline{RD}$ is asserted & 3EH is placed on the data bus.

At T_3:- fetch cycle is completed.

At T_4:- 8085 decodes the opcode & find outs that second byte needs to be read.
Here, contents of A_{15} - A_8 are unknown &
. data bus AD_7 - AD_0 goes into high impedance.

Step 2 :- After opcode fetch cycle, 8085 places 2001H on
T_1 { the address bus & increments PC to 2002H.
This second machine cycle M_2 is identified as
Memory read cycle ($IO/\overline{M} = 0, S_1 = 1, S_0 = 0$) & ALE is
asserted.

At T_2: $\overline{RD}$ is asserted & 32H is placed on the data bus.

At T_3: 8085 reads & stores the byte in accumulator.

Clock frequency f, = 2 MHz

T-state = clock period ($1/f$) = 0.5 μs

Execution time for opcode fetch = (4T) × 0.5 = 2 μs

" " " memory read = (3T) × 0.5 = 1.5 μs

" " " instruction = (7T) × 0.5 = 3.5 μs

Ques Draw the timing diagram of memory write m/c cycle.