* µP → Reprogrammable
µP → Embedded (Non-reprogrammable)

Bit:- Digit of binary number or code.

Nibble:- Group of 4 bits / 4 bit binary code.

Byte:- Group of 8 bits is called a byte.

Word size:- 16-bit binary no. or code.

Double word size:- 32 bit binary no. or code.

Multiple word size:- More than 32 bit binary no. or code.

Data:- Quantity operated by the instructions of a program.

Address:- Identification number of the memory location.

8085 µP has 16 bits for address for memory.

Memory word size:-

The memory word is the size of binary information that can be stored on a memory block/location.

8-bit data lines (8085 µP)

Microprocessor:-

Semiconductor device that can be programmed which fetches the instructions and data, decodes

and executes the instructions.

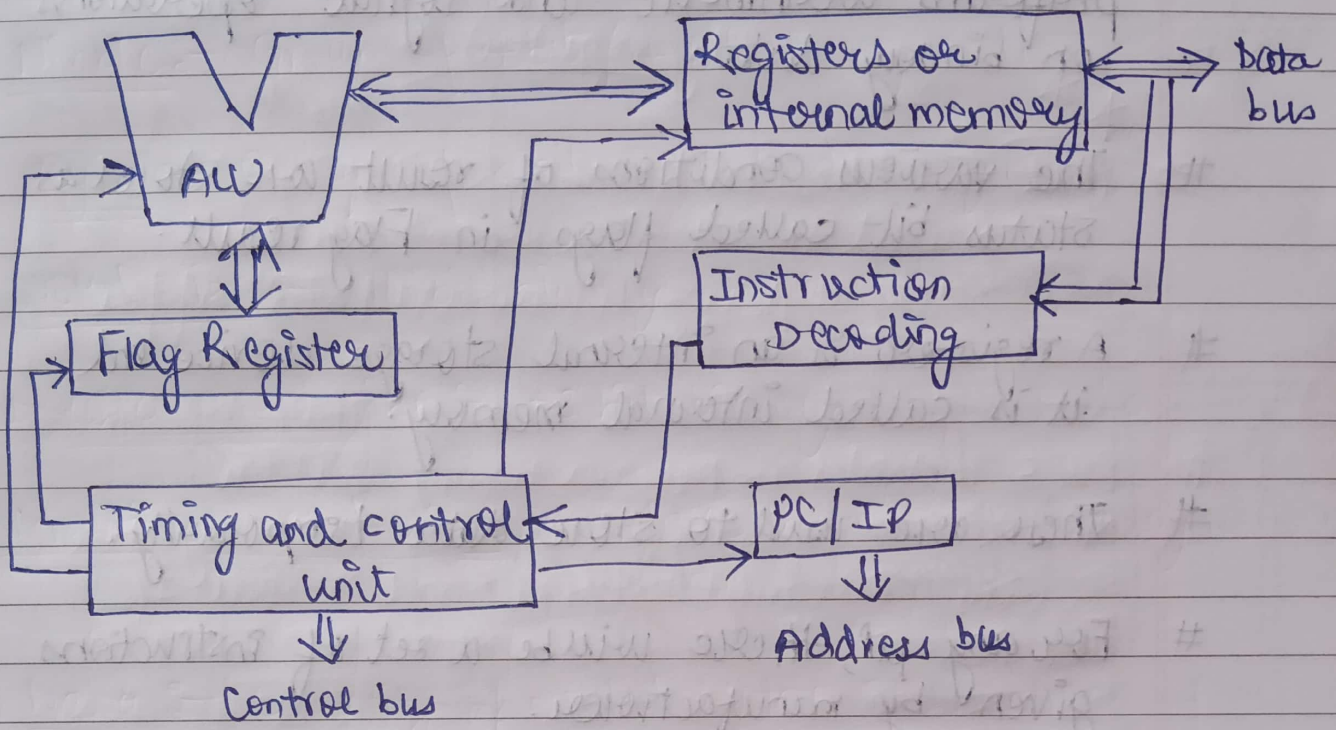
It is used as a CPU in computers. The basic functional blocks of µP are
→ ALU
→ Array of registers
→ CU.

The size of µP is identified by size of ALU.

Bus (group of conducting lines)
- → control signals → control bus
- → Data → Data bus
- → Address → Address bus

CPU Bus
Group of conducting lines that are directly connected to CPU/ µP. Signals are multiplexed.
- ↳ more than 1 signals are passed at different times.

Functional Building Blocks of 8085 µP



* IP → Instruction pointer.
* PC → Program counter (16-bits)

→ Flag ⇒ Various states of result are stored as status bit.

→ Types of Flags

① Sign (S) → MSB = 1 -ve
= 0 +ve

② Zero (Z) → Z = 1 (result = 0)
= 0 (result ≠ 0)

③ CY → carry flag (MSB se carry)

④ AC → Auxiliary carry flag (lower nibble se carry)

⑤ Parity Flag (P) → odd no. of 1s = 0
even no. of 1s = 1

ALU is the computational unit of µp which performs arithmetic and logical operations on binary data.

The various conditions of result are stored as status bit called flags in Flag result.

A register is an internal storage device and it is called internal memory.

These are used to store data temporarily.

For any µp, there will be a set of instructions given by manufacturer.
For doing any useful work with µp, we have to first write a program using these instructions and store them in a memory device external to the µp.

I/O or $\overline{M}$ = 0 I/O
= 1 memory

active low.

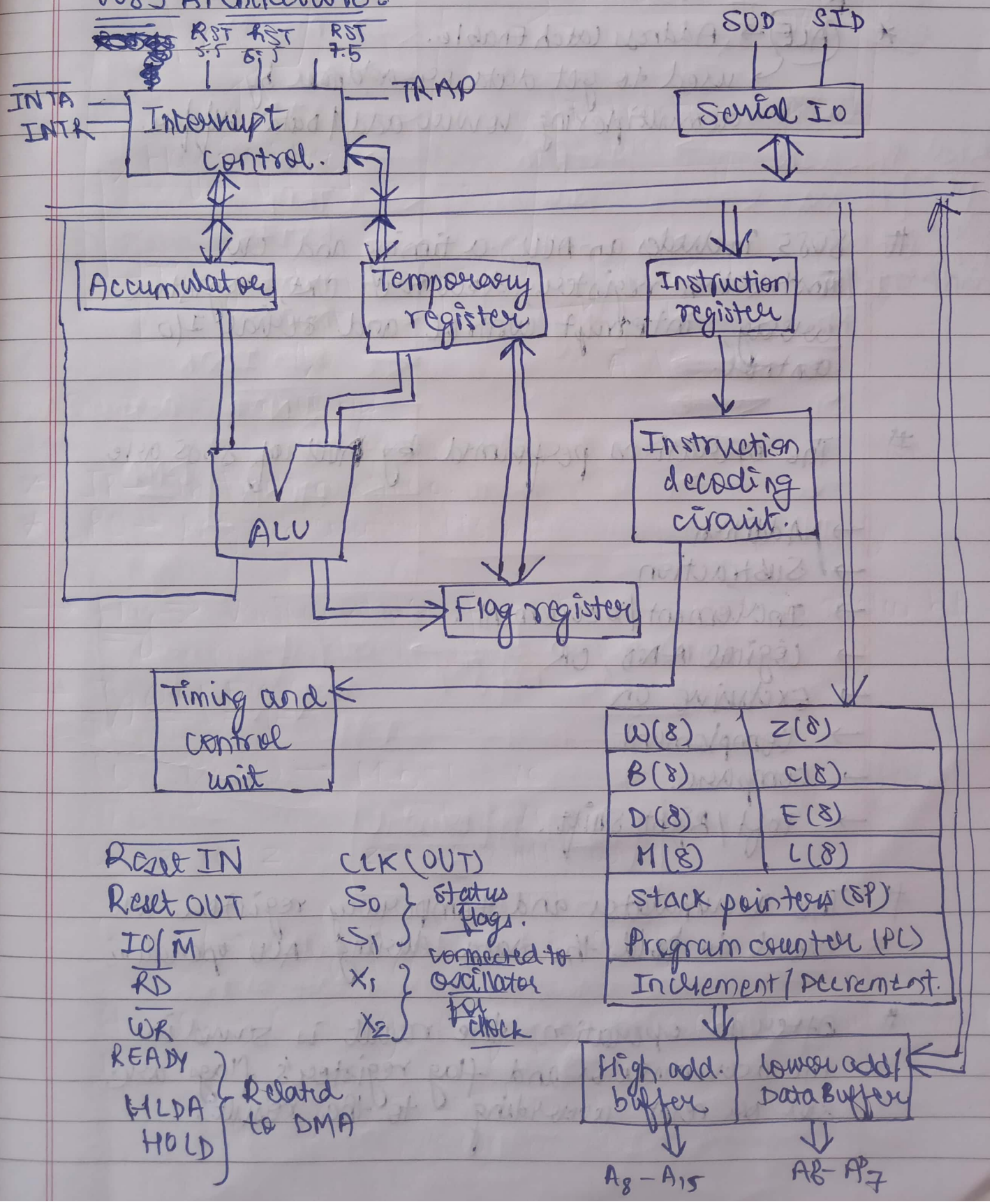
The IP generates the address of instructions to be fetched from the memory and sends it through the address bus to the memory.

The memory will send the instruction codes and data through the data bus. The instruction codes are decoded by the instruction decoding unit and it sends the information to the timing and CU.

invented in 1976 ← 8085 → 5V supply
8-bit

The CU will generate control signals for internal and external operations of the µP.

8085 Architecture :-



Reset IN
Reset OUT
IO/$\overline{M}$
$\overline{RD}$
$\overline{WR}$
READY
HLDA
HOLD } Related to DMA

CLK (OUT)
S_0, S_1 } Status flags.
X_1, X_2 } connected to oscillator for clock

S	Z	x	AC	x	P	x	CY

(Flag register)

* ALE → Address Latch Enable.
→ used to get data or address by demultiplexing lower add./data buffer.

8085 includes an ALU, a timing and CU, instruction register, decoder ckt, register array, interrupt control and serial I/O control.

The operations performed by ALU of 8085 are

→ Addition
→ Subtraction
→ Increment/Decrement.
→ Logical AND, OR
→ Exclusive OR
→ Complement
→ Compare.
→ Left/Right Shift.

The accumulator and temporary register are used to hold the data during ALU operations

After an operation the result is saved in the accumulator and flag register's flags are set or reset according to the result.

The accumulator and flag register are together called PSW (Program Status Word).

There are 5 flags in flag register:

① Sign Flag.
If the MSB is 1, then sign flag is 1.
If the MSB is 0, then it is 0

② Zero Flag.
If the result = 0, then zero flag is 1 and result ≠ 0 then zero flag is 0.

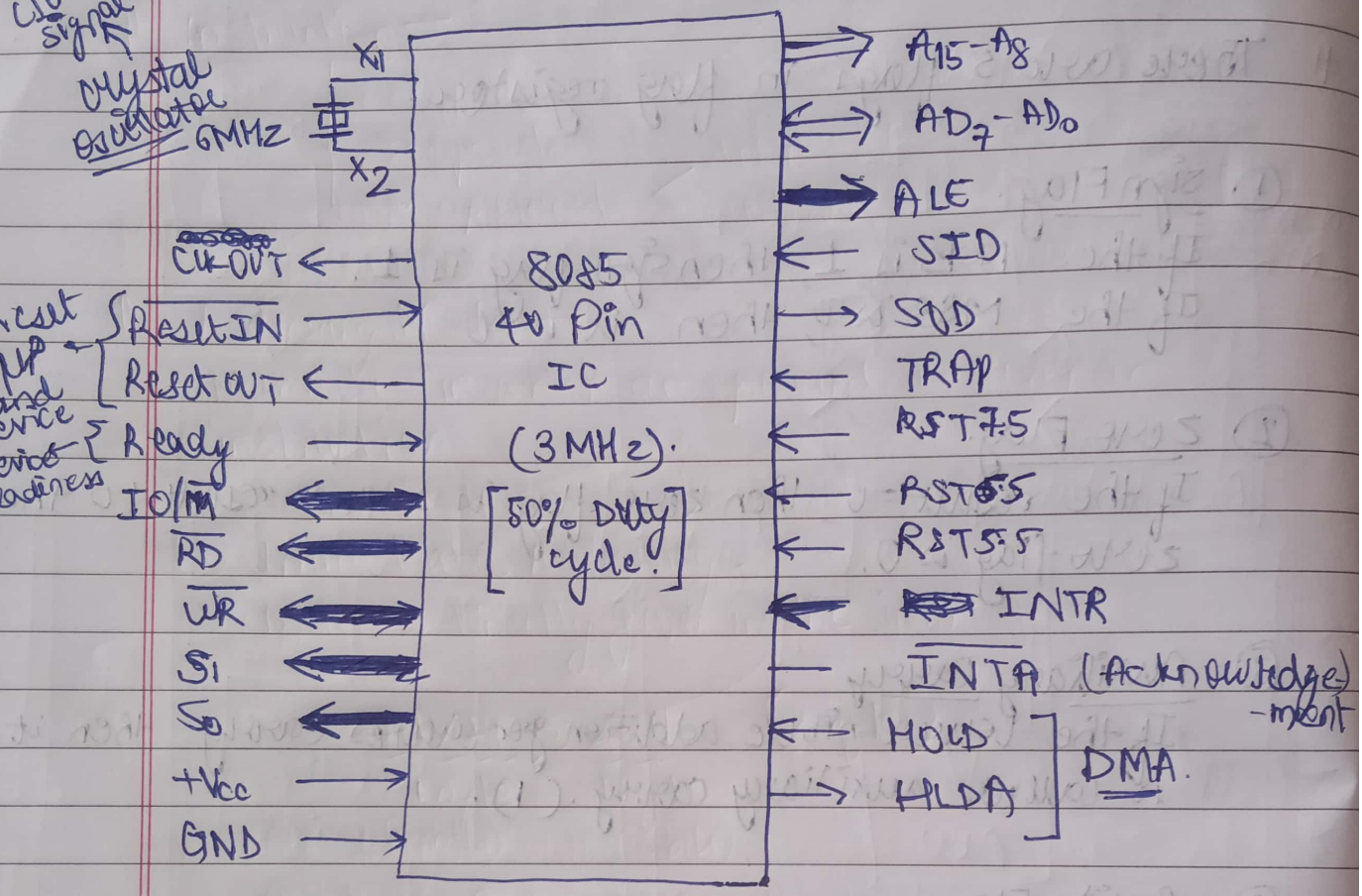
③ Auxiliary Carry
If the lower nibble addition generates carry, then it is called auxiliary carry. (1).

④ Parity Flag.
If odd no. of 1s in result, P = 0
even no. of 1s in result, P = 1.

⑤ Carry Flag.
If the addition of 2 nos. generates the carry after adding MSB, then carry flag is 1.

% Duty cycle = $\frac{T_{ON}}{T_{ON}+T_{OFF}}$

8085 µP Pin diagrams / assignments :-



Random Duty cycle → T Flip flop → 50% (frequency) duty cycle.

$$f_{clk} = \frac{f_{crystal}}{2}$$

★ Ready → It is used to tell whether other devices are ready to communicate or not.
eg. Printer (slow devices).

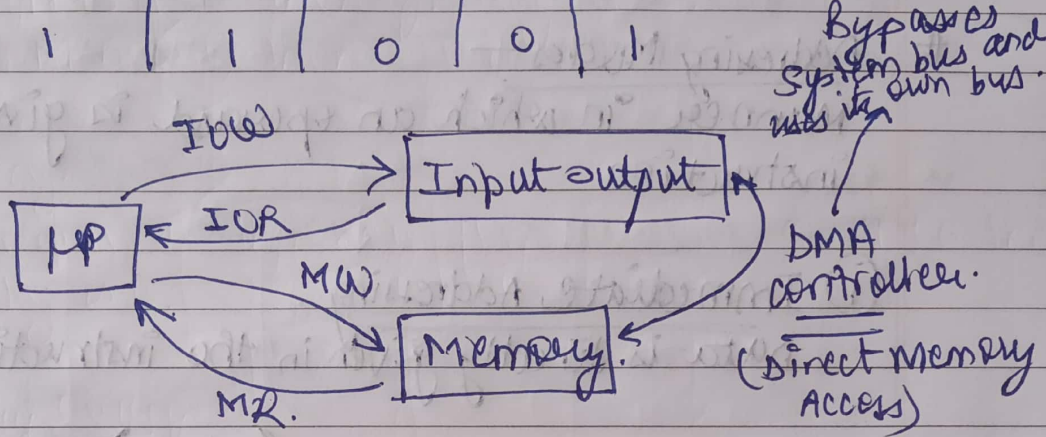
$S_0S_1 \rightarrow 00 \rightarrow$ Halt
$S_0S_1 \rightarrow 11 \rightarrow$ Fetch.

input signal to µP

[status signals] w.r.t. µP.

$S_0 \rightarrow$ output signal to µP

	IO/M	RD	WR	S_1	S_0
MR	0	0	1	1	0
MW	0	1	0	0	1
IOR	1	0	1	1	0
IOW	1	1	0	0	1



Status signals are defined for the µP as
S_1 : input w.r.t. µP
S_0 : out w.r.t. µP.

ALE → 1 → Address → stores in external latch (8 bit)
→ 0 → Data
(Address Latch Enable) H
to demultiplex AD_7-AD_0

SID, SOD → Serial communication (long communication).
distance.

TRAP
RST 7.5 6.5 5.5 INTR INTA → Interrupt.

8085 μp Addressing Modes :-

Instruction –

- opcode
- operand < data / address.

Addressing Modes –

Manner in which an operand is given in an instruction

① Immediate Addressing.

Data is directly given in the instruction.

MVI B, 25H (2 bytes)

1 byte (8 bits) | 1 byte (8 bits)

Additional:

```
  25 H
+ 3A H
-------
 15 H   = 5FH.
```

```
  38 H
+ 3A H
-------
 18 H
```

72 H

load

(3 bytes)

eg. LXI B, 2000H

register pair → { BC, DE, HL }

(2) Register Addressing Mode

Data is given in a register.

MOV B, C ; 1 byte. (B ← C)

opcode

INR B ; 1 byte.

opcode. (B ← B+1)

INX B ; 1 byte.

(BC ← BC+1).

eg. A=0, B=0, C=0

```
MVI A, 00H          MVI A, 00H
MVI B, 00H    Or.   MOV B, A
MVI C, 00H          MOV C, A.
```

(6 bytes) easy.

memory efficient (4 bytes)

or

```
SUB A
MOV B, A
MOV C, A
```

most memory efficient. (3 bytes.)

(3) Direct Addressing Mode.

~~Data~~ Address is given in instruction.

LDA 2000H ;

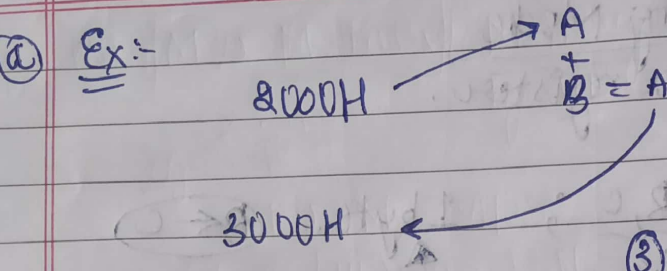
opcode. (1 byte) → address (2 bytes)

[Load accumulator]

STA 2000H ;

Store accumulator. (1 byte) → address (2 bytes)

(a) Ex:-

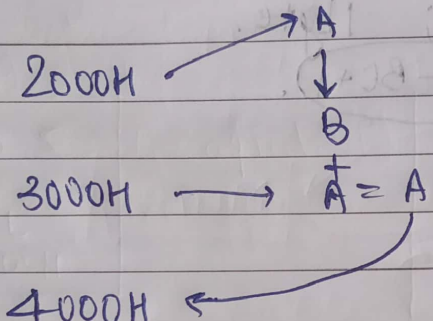


MVI B, 20H ← LDA 2000H;
ADD B;
STA 3000H;

(3) LDA 2000H
(2) MVI B, 20H
Or. (1) ADD B
(3) STA 3000H

9 bytes

(b) Ex:-



(3) LDA 2000H;
(1) MOV B, A;
(3) LDA 3000H;
(1) ADD B;
(3) STA 4000H;

11 bytes.

(4) Indirect Addressing Mode.

Address is given in register pair (BC, HL, DE).

~~LDAX B 2000H~~

LDAX B; 1byte
STAX B; 1byte.

* Most of the operations are performed using indirect addressing mode.

(5) Implied Addressing Mode:
operand is not at all specified in the instruction either directly or indirectly.

STC	Set carry flag
NOP	No operation
CMC	Complement carry flag
HLT	Halt.

Instruction Set of 8085 µP :-

(1) Data Transfer Groups Instructions

This group deals with the transfer of data from one place to another place.

(a) MVI R, Data (8-bit) → 7 opcodes (2-bytes)
MVI A, ...
MVI B, ...
⋮
eg.
MVI A, 32H;

(b) LXI Rp, 16-bit data → 3 opcodes (3 bytes)
BC
DE
HL
eg.
LXI B, 2000H
BC ← 2000H (B ← 20H, C ← 00H)

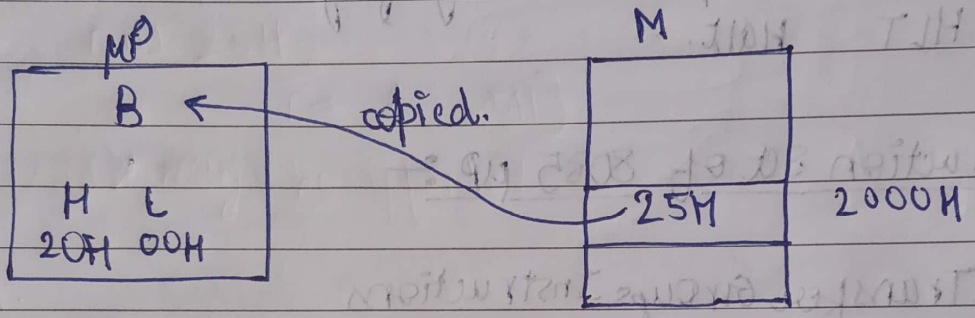
(c) MOV RD, RS → 49 opcodes (1 byte).
A, B
A, C.
⋮
eg. MOV A, B;
A ← B
25H 25H (remain same)

(d) MVI M, Data (2 byte) [Memory location determined by HL register pair] → 1 opcode

eg.
MVI M, 25H ;
[HL] ← 25H.

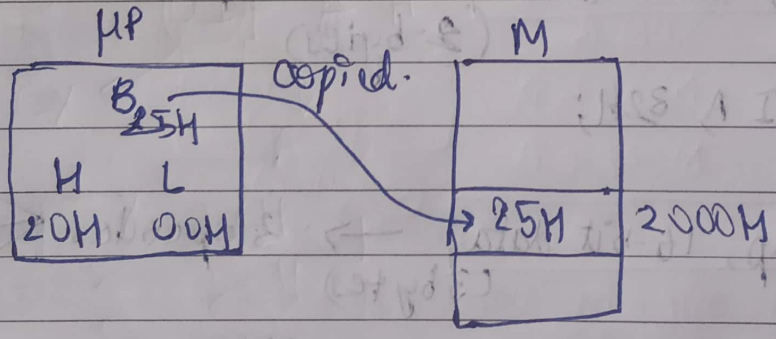
(e) MOV R, M (1 byte) → 5 opcodes

eg.
MOV B, M.



(f) MOV M, R (1 byte) → 5 opcodes

eg.
MOV M, B



(g) LDA Address (3 bytes) → 1 opcode.

eg.
LDA 2000H;

(h) STA Address (3 bytes) → 1 opcode.

eg. STA 2000H;

(i) LDAX B; (1 byte) → 2 opcodes.
A ← [BC]

NOTE:→ LDAX H; is not a valid instruction as it is same as MOV A,M

(j) STAX B; (1 byte) → 2 opcodes
[BC] ← A

(k) LHLD 2000H; (3 bytes) → 1 opcode.
L ← [2000H]
H ← [2001H]

* loads HL register pair with the data on memory location 2000H.

(l) SHLD 3000H; (3 bytes) → 1 opcode.
[3000H] ← L
[3001H] ← H

(p) XTHL;
exchange the top with HL;
SP ↔ HL.

(m) PCHL;
PC ← HL.

(n) SPHL;
SP ← HL

(o) XCHG;
HL ↔ DE.
H ↔ D
L ↔ E

(3) Arithmetic Instruction set

(a) **ADD R;** ⟶ 6 opcodes.

$A \leftarrow A + B$

CY may be affected.

(b) **ADD M;**

$A \leftarrow A + [M]$

CY may be affected.

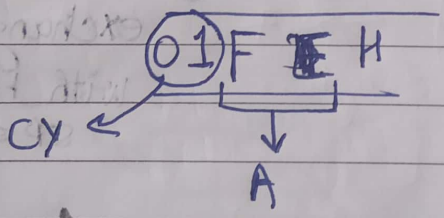
(c) **ADI 25H.**

$A \leftarrow A + 25H$

CY may be affected.

\# ex:

```
  FFH
+ FFH
------
 30 30 H
```



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

(d) **SUB B;**

$A \leftarrow A - B$

(e) **SUB M;**

$A \leftarrow A - [M]$.

(f) **SUI 25H;**

$A \leftarrow A - 25H$

0011 0001
1100 1111
1101 1101
(1) 1010 1100
0101 0100
54

ex:

(1) (19)
~~23~~ 3H
− 15H
0EH

23H
0010 0011

15H
0001 0101
1's 1110 1010
2's 1110 1011

0010 0011
+ 1110 1011
(1) 0000 1110

$\overline{CY}$ ← (1)

0EH

$\overline{CY} = 1$
P = 1
S = 0
AC = 0
Z = 0.

23H → 0010 0011 —2's→ 1101 1101
15H → 0001 0101 —2's→ 1110 1011

0010 0011
1110 1011
(1) 0000 1110

1101 1101
1110 1011
(1) 1100 1000
0011 1000

Ex: (a) 23H + 31H = 54H P=0 S=0 Z=0 AC=0 CY=0

0101 0100

①

(b) 37H + 29H = 60H P=1 S=0 Z=0 CY=0 AC=1

0011 0111
+ 0010 1001
0110 0000

60H

(c) -28H -31H -54H S=1 P=1 Z=0 AC=1 CY=1

① → AC

1101 1101
+ 1100 1111
① 1010 1100 → 1101 0100 (2's)

CY → ACH. (Accumulator.)

-54H

(g) ADC R;

A ← A+R+CY

Used to add 2 16 bit numbers.

(h) ADC M;

A ← A+[M]+CY

(i) ACI 25H
A ← A + 25H + CY.

(j) SBB M
A ← A - [M] - CY

(k) SBI 25H
A ← A - 25H - CY

(l) SBB R;
A ← A - R - CY.

(m) INR R; [Affects flags]
R ← R + 1

(n) INX Rp;
Rp ← Rp + 1
[Not a part of ALU part of incremental and decremental ckt.]
[Does not affect any flag]

(o) INR M;
[M] ← [M] + 1

(p) DCR R;

(q) DCX Rp;

(r) DCR M;

(s) DAD D;
Double addition.
HL ← HL + DE

(t) DAD B;
HL ← HL + BC

(u) DAD H;
HL ← HL + HL

(v) DAA;
Decimal Adjust After Addition.

ex:
```
  45H
+ 45H
------
  8AH  → Not valid BCD but valid Hex.
+ 06H
------
  90H  → Now valid BCD.
```

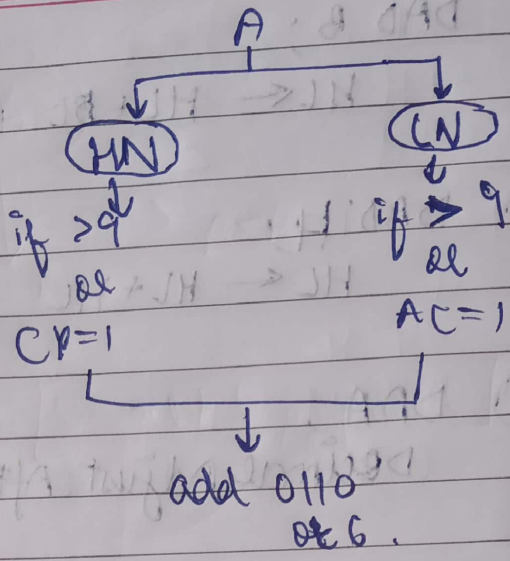
ex:
```
  25H
+ 25H
------
  4AH  Not valid BCD
+ 06H
------
  50H  Valid BCD
```

AC=0

```
  0010  0101
  0010  0101
-------------
  0100  1010
        0110
-------------
  0101  0000
```

ex:
```
  60H
  60H
------
  C0H  Invalid BCD
+ 60H
------
CY ← 120  BCD
```

ex:
$$\begin{array}{r} 99\ H \\ +\ 99\ H \\ \hline 132\ H \\ +\ 66\ H \\ \hline 198\ H \end{array}$$



(3) Logical Instruction set.

(a) ANA R;
A ← A ∧ R

(b) ANA M;
A ← A ∧ [M].

(c) ANI 25H;
A ← A ∧ 25H.

(d) ORA R;
A ← A ∨ R;

(e) ORA M;
A ← A ∨ [M]

(f) ORI 25H;
A ← A ∨ 25H.

(g) XRA R;
A ← A ⊕ R

(h) XRA M;

$A \leftarrow A \oplus [M]$

(i) XRI 25H

$A \leftarrow A \oplus 25H$

Q) Assume A = 35H.

1) Clear the lower nibble of A.

A = 35H = 0011 ; 0101 ANI F0H;

(AND) 1111 ; 0000

Required = 0011 ; 0000 → 30H

or.

A = 35H = 0011 ; 0101 XRI 05H;

(XOR) 0000 ; 0101

Required = 0011 ; 0000 → 30H

2) Clear the higher nibble of A.

ANI 0FH

XRI 30H.

3) Set the lower nibble of A.

ORI 0FH

XRI 0AH

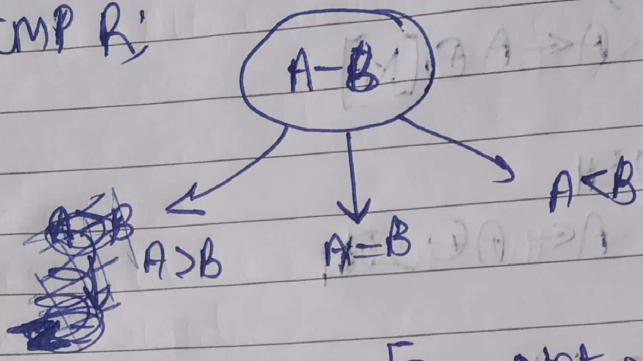
4) Complement the lower nibble of A.

0011 0101 XRI 0FH

⊕ 0000 1111

0011 1010 → 3AH

(j) CMP R;



	CY	Z	[S → Not required]
A>B	0	0	
A=B	0	1	
A<B	1	0	

(k) CMP M;

[A-(M)]

(l) CPI 25H;

[A-25H]

(m) STC → Set the carry flag. 1 ← CY

(n) CMC → Complement carry flag. $\overline{CY} \leftarrow CY$.

(o) CMA;

$\overline{A} \leftarrow A$

NAND

ANA R;
CMA;

XNOR

XRA R;
CMA;

NOR

ORA R;
CMA;

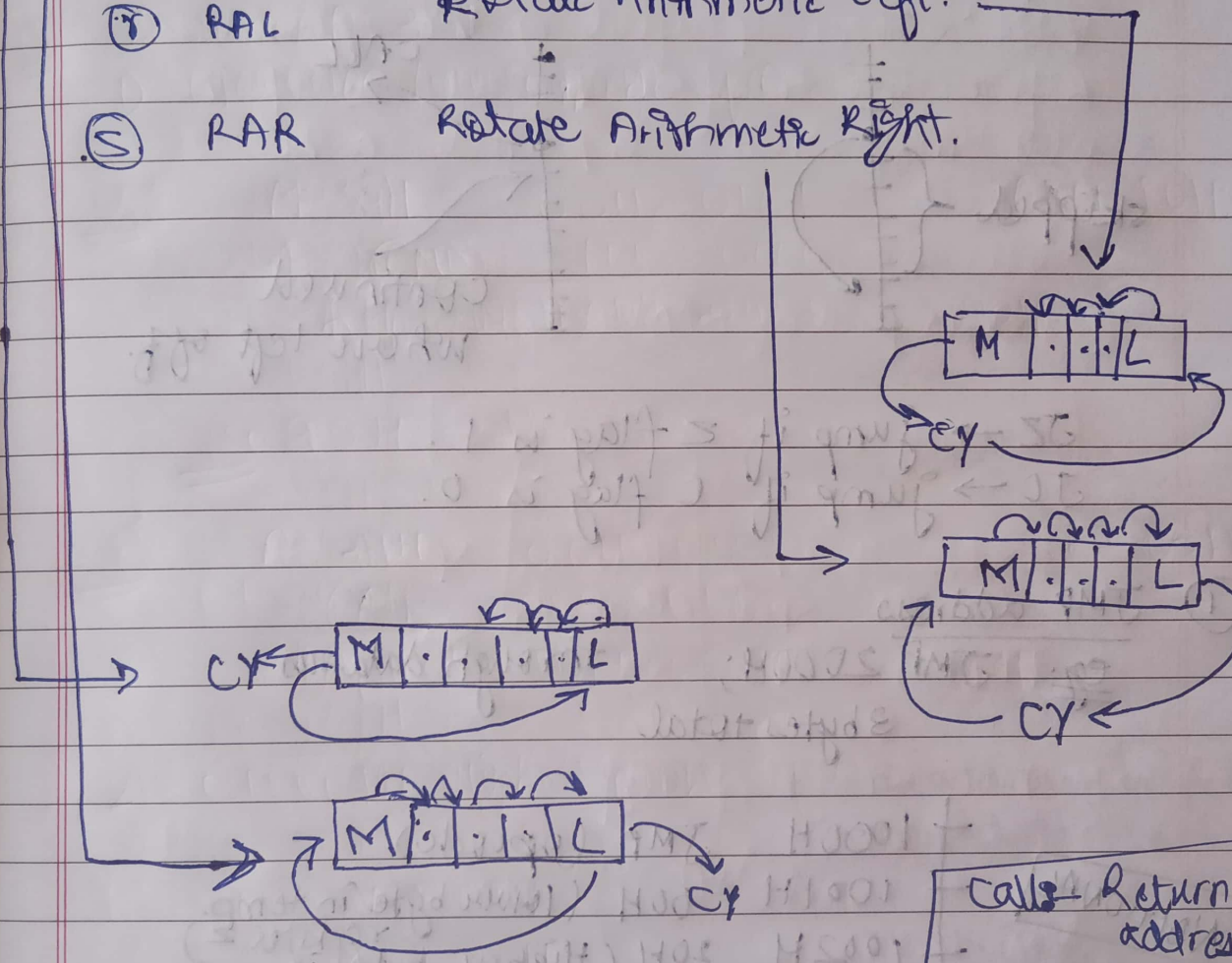
* Rotating Instruction.

(p) RLC — Rotate left carry

(q) RRC — Rotate Right carry

(r) RAL — Rotate Arithmetic Left.

(s) RAR — Rotate Arithmetic Right.



Call :- Return address req.

Jump :- No return address req.

Q) Multiply by 2 :

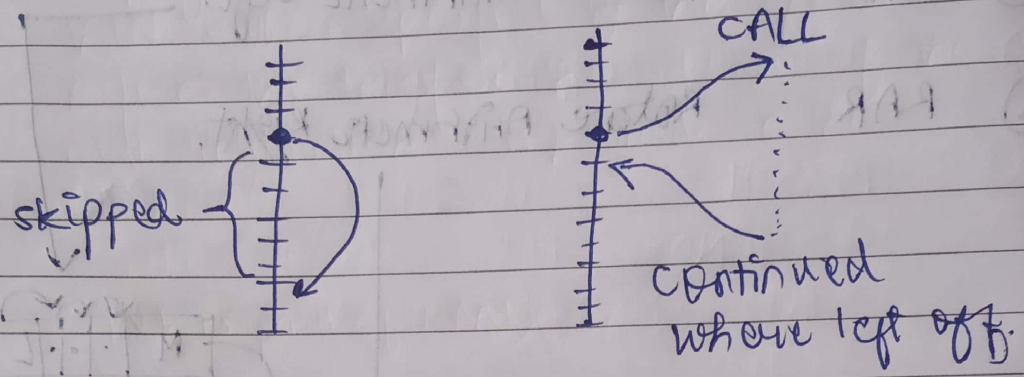
0000 0011

0000 0110

JUMP v/s CALL instruction

unconditional
conditional (Based on flags)

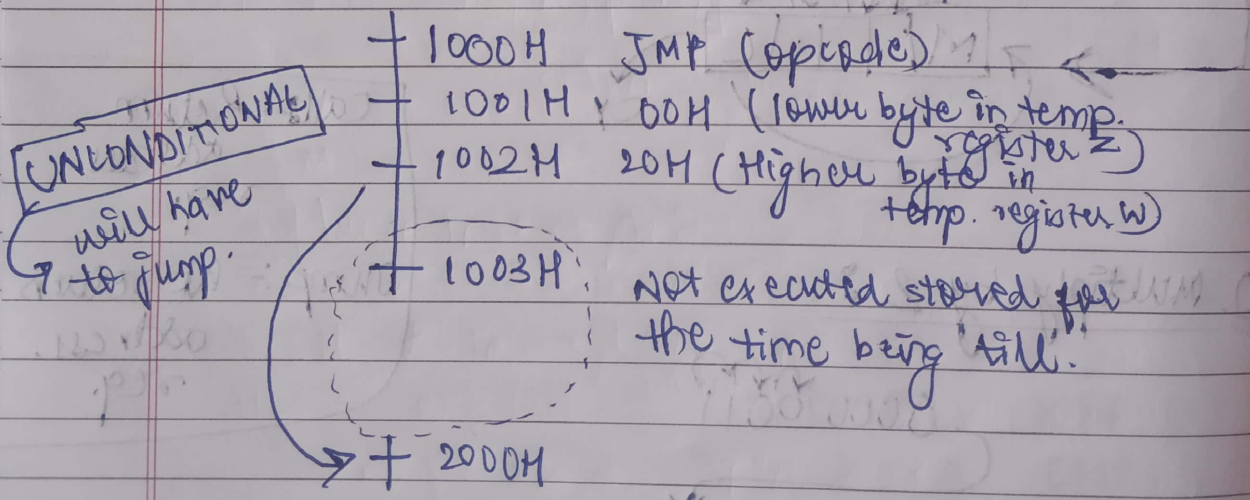
return address not required
stack not used.



JZ → jump if z flag is 1.
JC → jump if C flag is 0.

① JMP address

~~eg.~~ JMP 2000H; through databus
3 bytes total



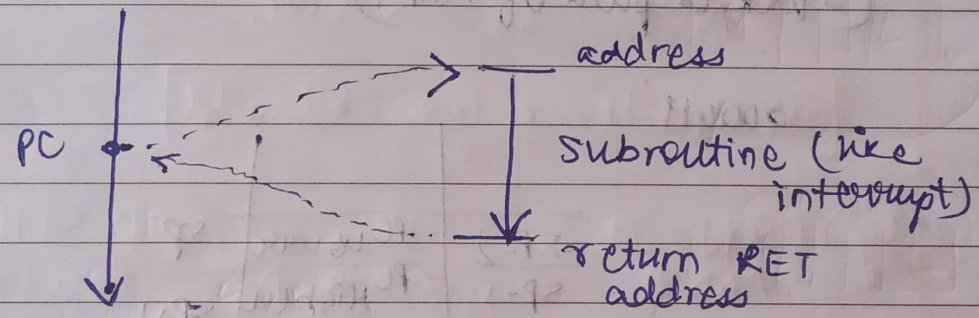
Labels can be used to specify address.

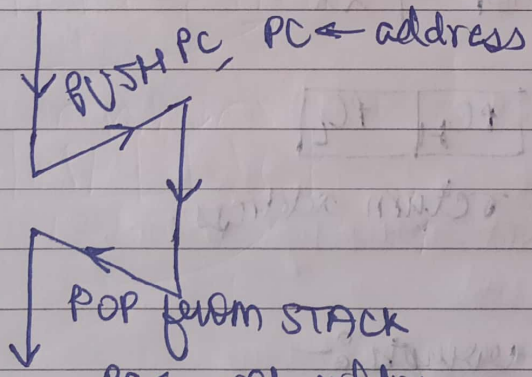
PC ← 2000H (WZ)

JC
JNC
JM ; jump when sign bit = 1 (-ve numbers)
JP ; sign bit = 0 (+ve numbers)
JPE ; parity even ; P = 0
JPO ; parity odd ; P = 1
JZ ; Z = 1
JNZ ; Z = 0.

CONDITIONAL → flags used.

② CALL Address





PC address will be stored in stack

stack Pointer SP (top address of stack)

CC
CNC
~~JNC~~ JCN
CP
CZ
CNZ
CPE
CPO

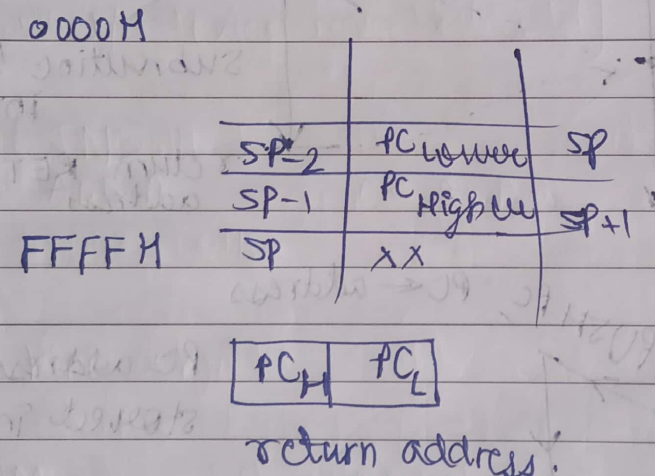
return → conditional
return → unconditional.

CONDITIONAL

Unconditional call:
No flags are affected after the execution of jump and call instruction.

Conditional call:
A program sequence is transferred to the address specified by the operand. Before the transfer the address of the next instruction is pushed on the stack.

(Stack is part of RAM)



Software Interrupts:-

reset RST_n → have same address

n×8

$n = \{0, 1, 2, 3, \ldots, 7\}$

0×8	0000H
1×8	0008H
2×8	0010H
3×8	0018H

standard sub routines.

8 software + 5 hardware
= 13 total 8085 µp interrupts.

Q) Ret v/s CALL
↙ 1byte ↘ 3byte.

Stack, I/O and Machine Control

1) PUSH Rp eg. PUSH B → push data in reg. pair BC onto stack.

BC
High low

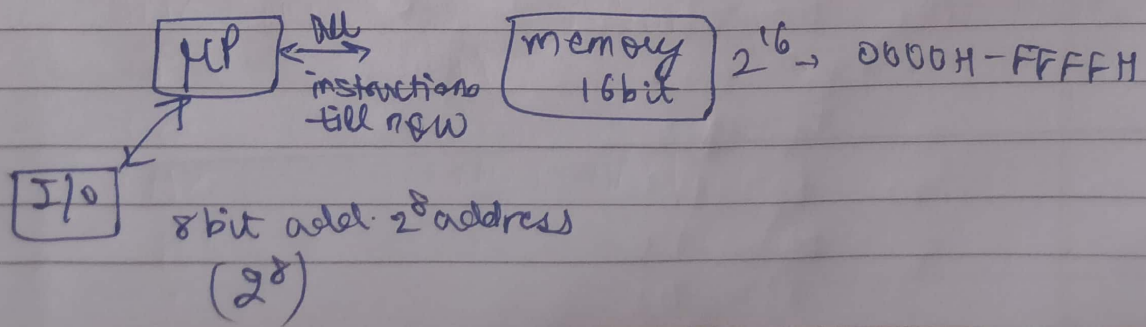
SP-2	C
SP-1	B
SP	XX

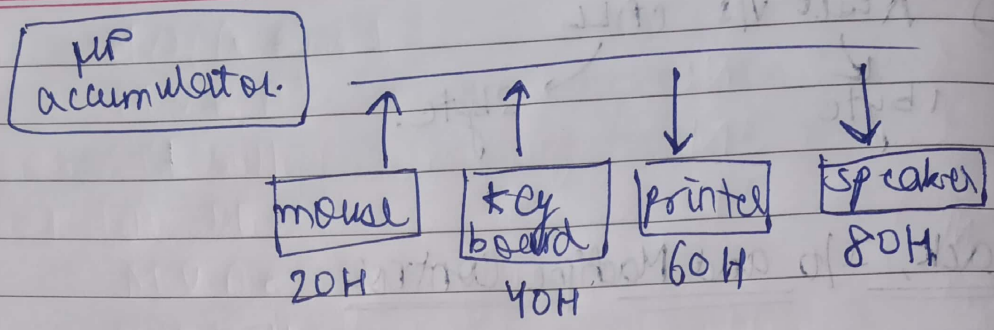
SP ← SP-1
[SP] ← B
SP ← SP-1
[SP] ← C.

2) POP Rp eg. POP B

C ← [SP]
SP ← SP+1
B ← [SP]
SP ← SP+1

I/O Related ← IN address
OUT address.





IN 20H

Data from 20H → Accumulator.

OUT 60H

Accumulator → 80H.

[all I/O device have 8bit add.]

eg:) IN 20H
ADD A (A ← A+A)
OUT 60H

→ Machine control.

1	SIM	Set interrupt mask.
2	RIM	read " "
3	EI	Enable interrupt.
4	DI	Disable "
5	NOP	No operation (if delay needed)
6	HLT	Halt.

Q) Assume CY = 0.

MVI A, 07H	A = 0000 0111
RLC	A = 0000 1110
MOV B, A	B = A
RLC	A = 0001 1100
RLC	A = 0011 1000
ADD B	A = 01000110
RRC	A = 0010 0011 = 23H

S can't decide if no. is +ve or -ve (CY has to be checked)
if S=1 CY=1 then its -ve
if S=0 CY=0 then its +ve (if borrow needed then -ve if not needed then result +ve).

Q)

```
LXI H, 2000H
LDA 2002H
XRA M
MOV E, A
MVI D, 20H
LDAX D
OUT 01H
HLT
```

Memory	content
2000H	00H
2001H	01H
2002H	02H
2003H	03H

Soln

H	L
20H	00H

A = 02H
⊕ 00H
A = 02H

E = 02H
D = 20H

A ← [DE]
A = 02H.

Q)

```
MVI A, Byte 1
MOV B, A
SUI 50H
JC DEL
MOV A, B
SUI 80H
JC DISP
DEL: XRA A
     OUT Port 1
     HLT
DISP: MOV A, B
      OUT Port 2
```

What will display on Port 2?

Soln →

A ← (Byte 1 - 50H)

B ← Byte 1 (N ≥ 50H, CY=0)

N < 50H CY=1
Port 1 ← 00H

A ← (Byte 1 - 80H)

A ← Byte 1 → Port 2.

N < 80H
CY=1

~~N ≥ 80H~~
~~Port 1 ← 00H~~

B = 000
C = 001
D = 010
E = 011
H = 100
L = 101
M = 110
A = 111

(Values of register)

Hexcode.

ADD → 10000

MOV → 01

Hexcode for MOV A, M:

01 111 110

7EH.

Hexcode for ADD B;

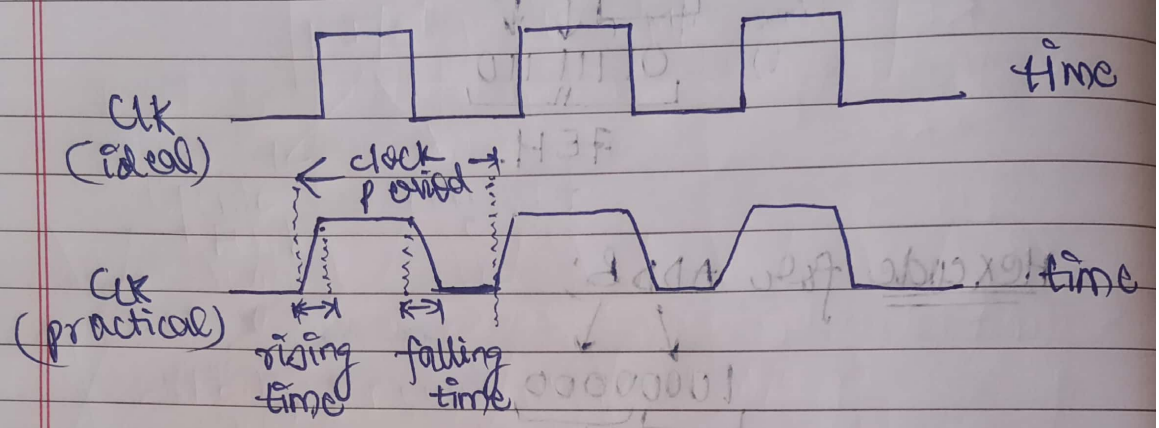
10000000

80H

Unit-2.

Assembly Language Programming and Timing Diagram.

(1) T-state.



1 T-state = 1 clock period.

It is a time in which the µP completes a subpart of an instruction in a single clock cycle.

(2) Machine cycle.

→ to complete one whole operation.

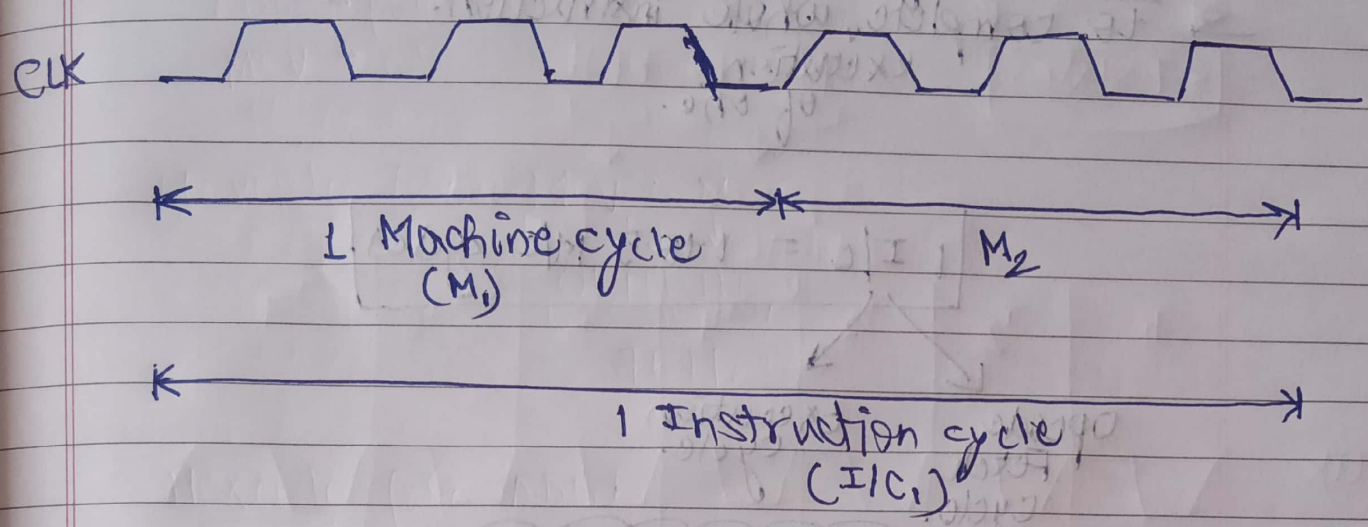
1 M/c cycle = 3 to 6 T-States.

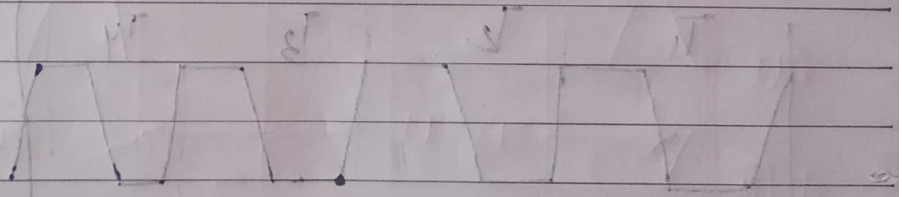
→ 9 M/c cycles ⟹

(a) Opcode fetch
(b) Memory read
(c) " write
(d) I/O read
(e) " write.
(f) Interrupt acknowledge
(g) Halt.
(h) Hold
(i) Reset.

(b) to (e) — Imp.

Timing Diagram.





③ Instruction cycle.

→ to complete whole instruction execution of one.

1 I/C = 1 to 5 M/C cycles

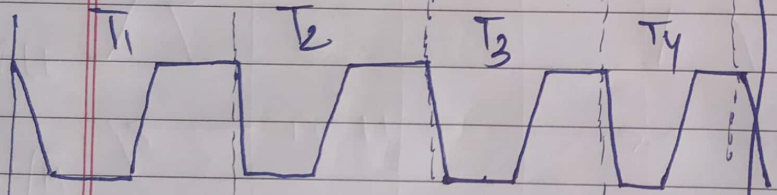
- Opcode Fetch cycle.
- Execution cycle.

Instructions 1 byte generally takes 1 M/C

ADDB;

Opcode Fetch Cycle :-

4 T-states



Places add. of memory location ALE = 1

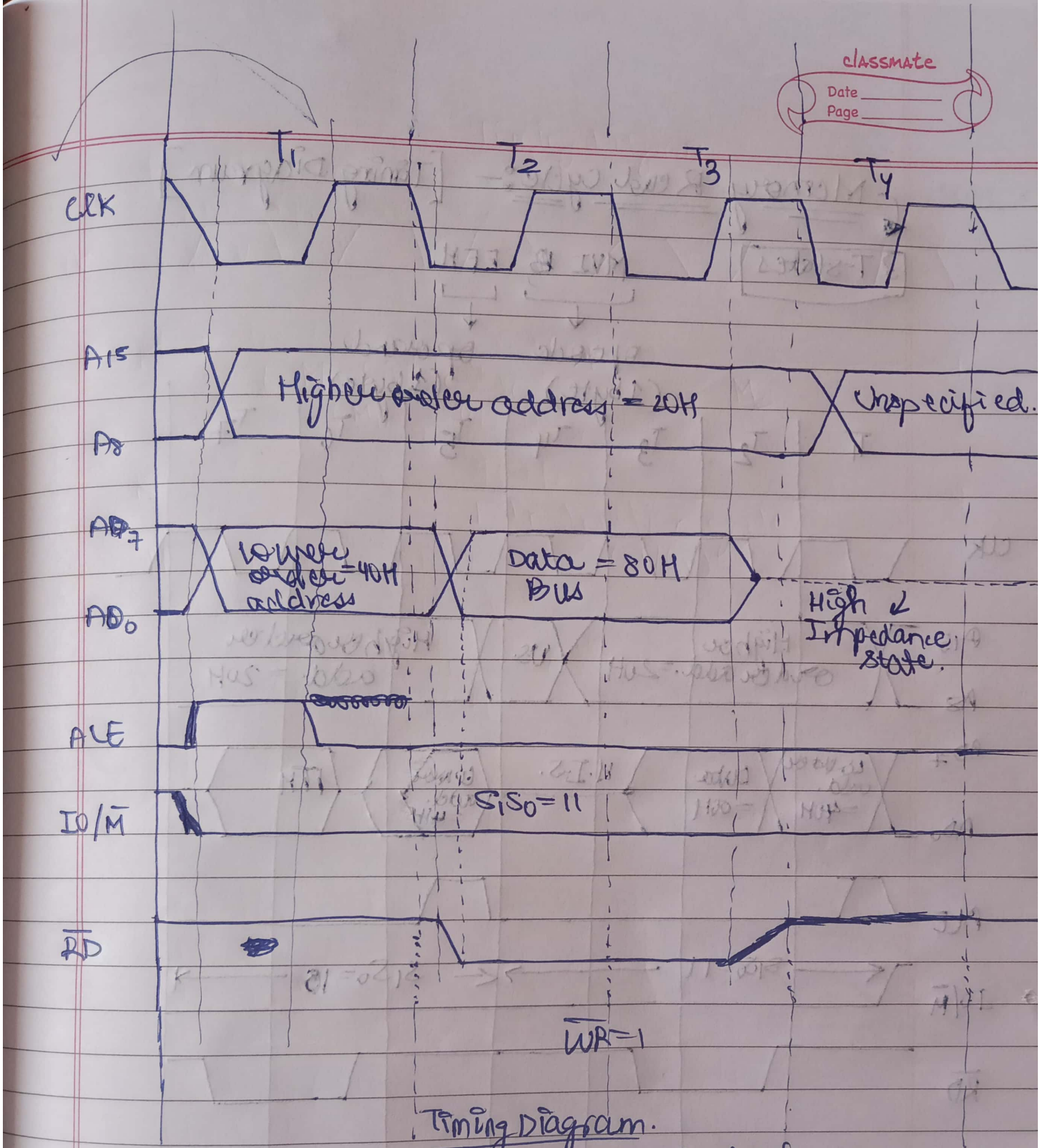
Get the code from memory $\overline{RD}$ = 0

Transfer opcode to MP $\overline{RD}$ = 0

Decode the opcode

Timing Diagram includes —

→ clock
→ Higher order Add A_{15}-A_8
→ Multiplexed Add./Data bus AD_7-A
→ ALE
→ IO/$\overline{M}$ and S_1S_0
→ control s/g $\overline{RD}$ and $\overline{WR}$.



Timing Diagram for opcode fetch cycle.

Most of the 1byte instructions only have opcode fetch cycles.

Memory Read cycle:- [Timing Diagram]

3 T-states

MVI B, FFH

MVI B → opcode (1 byte)

FFH → operand (1 byte)