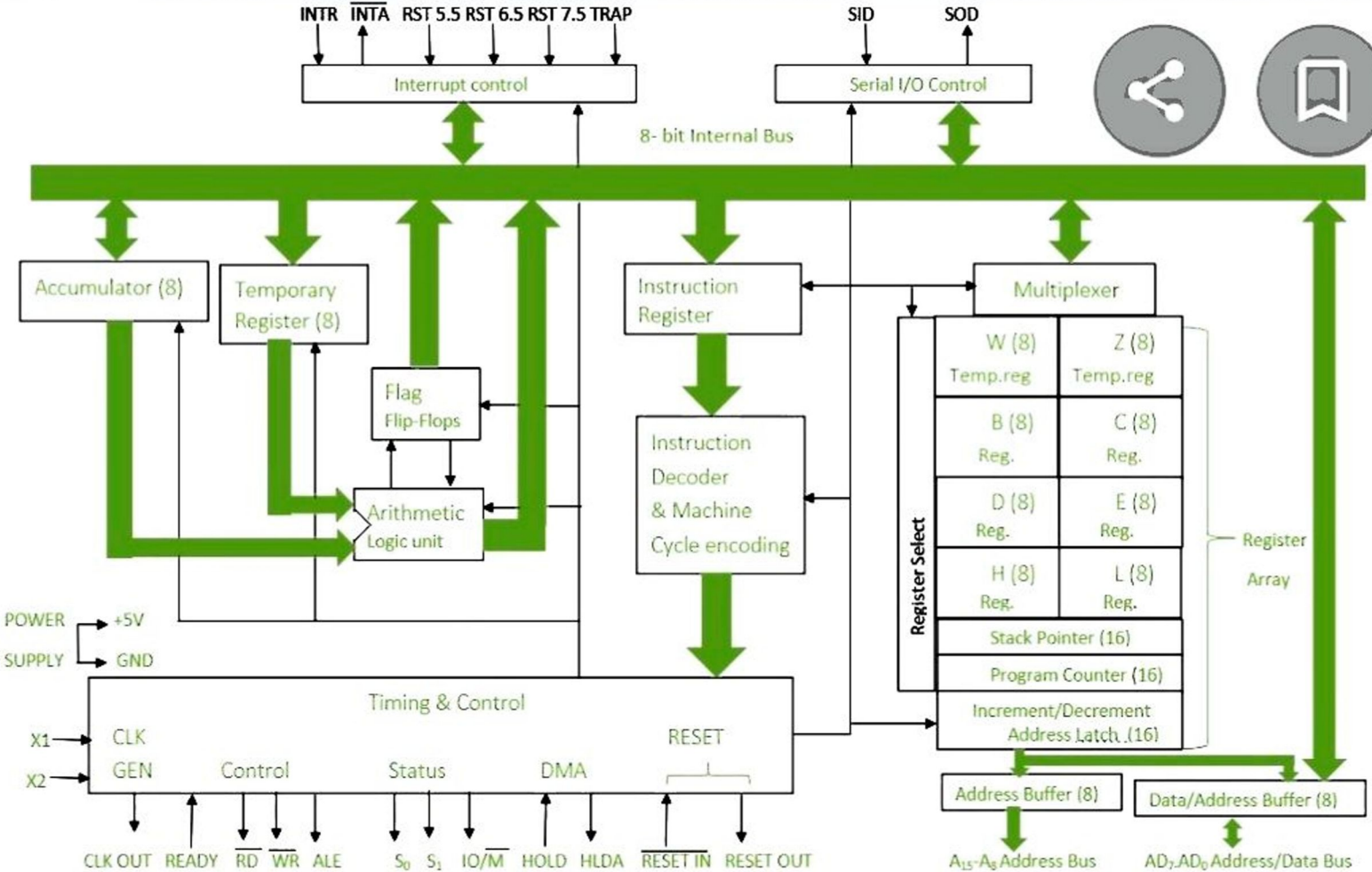
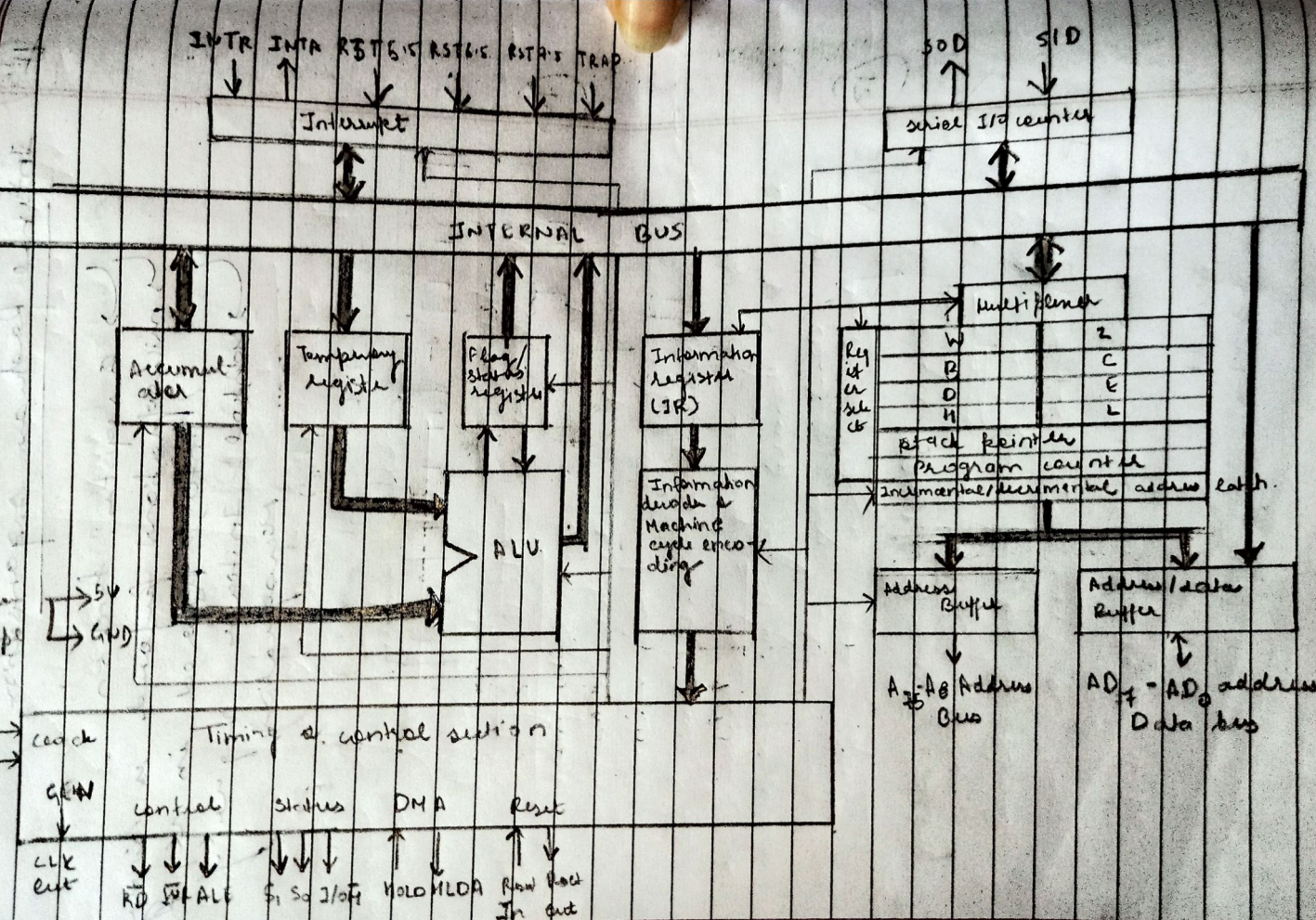
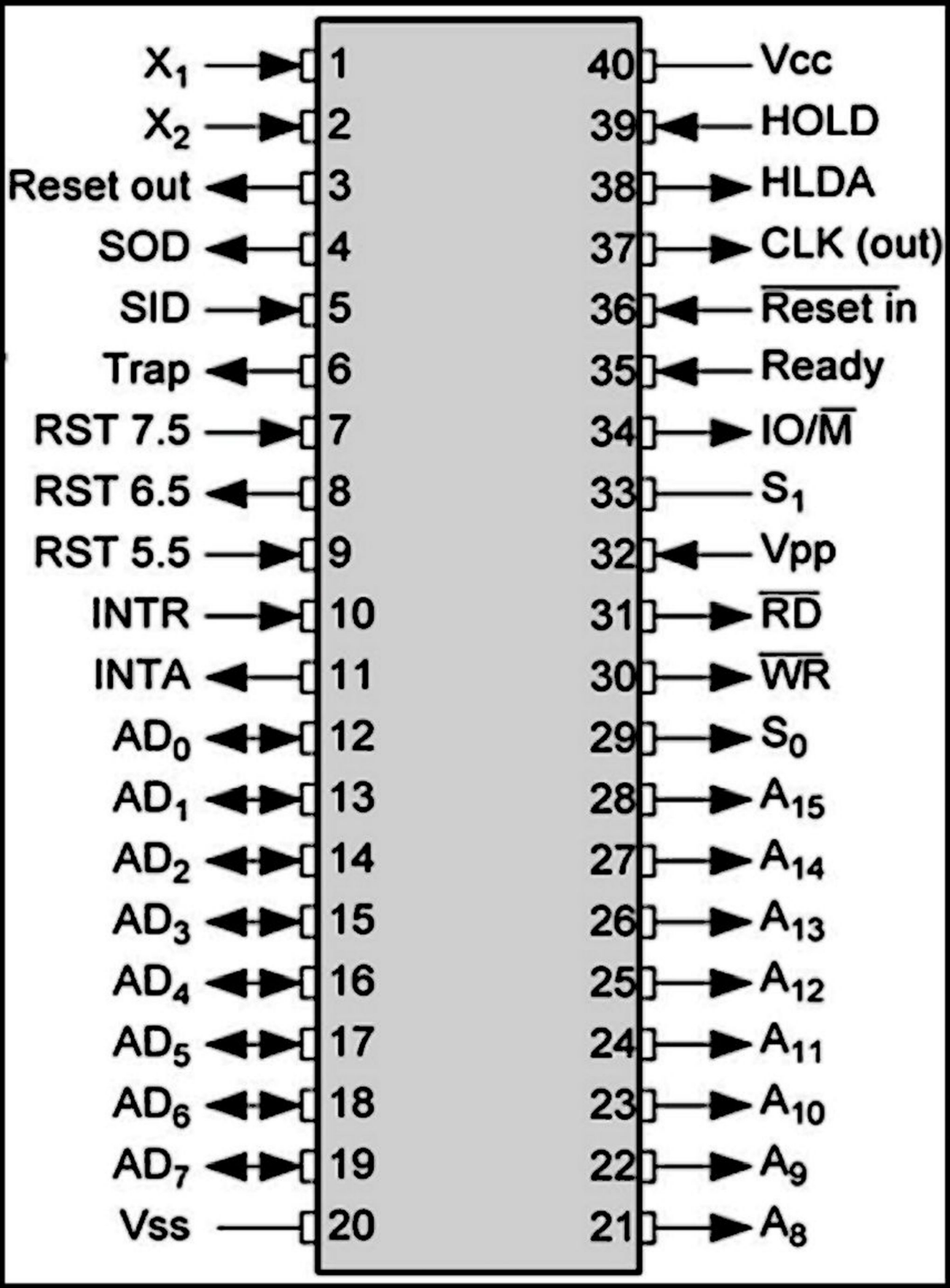
Block diagram 8085


INTR
INTA
RST 5.5 RST 6.5 RST 7.5 TRAP
SID
SOD
Interrupt control
Serial I/O Control
8- bit Internal Bus
Accumulator (8)
Temporary Register (8)
Flag Flip-Flops
Arithmetic Logic unit
Instruction Register
Instruction Decoder & Machine Cycle encoding
Multiplexer
Register Select
W (8) Temp.reg
Z (8) Temp.reg
B (8) Reg.
C (8) Reg.
D (8) Reg.
E (8) Reg.
H (8) Reg.
L (8) Reg.
Register Array
Stack Pointer (16)
Program Counter (16)
Increment/Decrement Address Latch (16)
POWER SUPPLY
+5V
GND
Timing & Control
X1
X2
CLK GEN
Control
Status
DMA
RESET
Address Buffer (8)
Data/Address Buffer (8)
CLK OUT
READY
RD
WR
ALE
S0
S1
IO/M
HOLD
HLDA
RESET IN
RESET OUT
A15-A8 Address Bus
AD7-AD0 Address/Data Bus

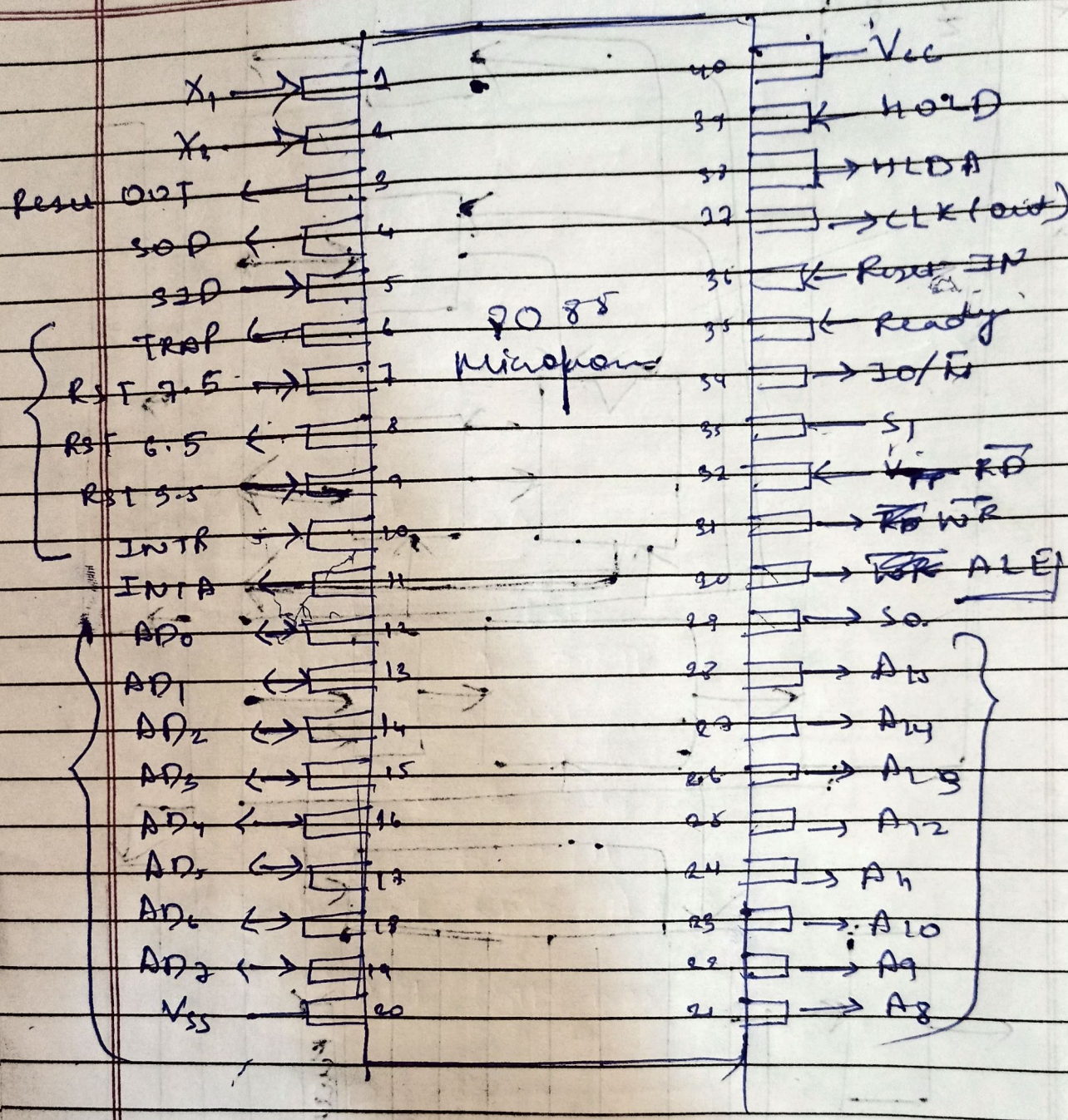
8085 Microprocessor Architecture



Pin diagram 8085


X1 1
X2 2
Reset out 3
SOD 4
SID 5
Trap 6
RST 7.5 7
RST 6.5 8
RST 5.5 9
INTR 10
INTA 11
AD0 12
AD1 13
AD2 14
AD3 15
AD4 16
AD5 17
AD6 18
AD7 19
Vss 20
40 Vcc
39 HOLD
38 HLDA
37 CLK (out)
36 Reset in
35 Ready
34 IO/M
33 S1
32 Vpp
31 RD
30 WR
29 S0
28 A15
27 A14
26 A13
25 A12
24 A11
23 A10
22 A9
21 A8

Page No.
Date :



(1) clock → In

(2) ~~Internally Initiated signal~~ Reset → out

(3) Serial I/O ~~lines~~ control → O → out
I → In

(4) Interrupt → ~~out~~ activate

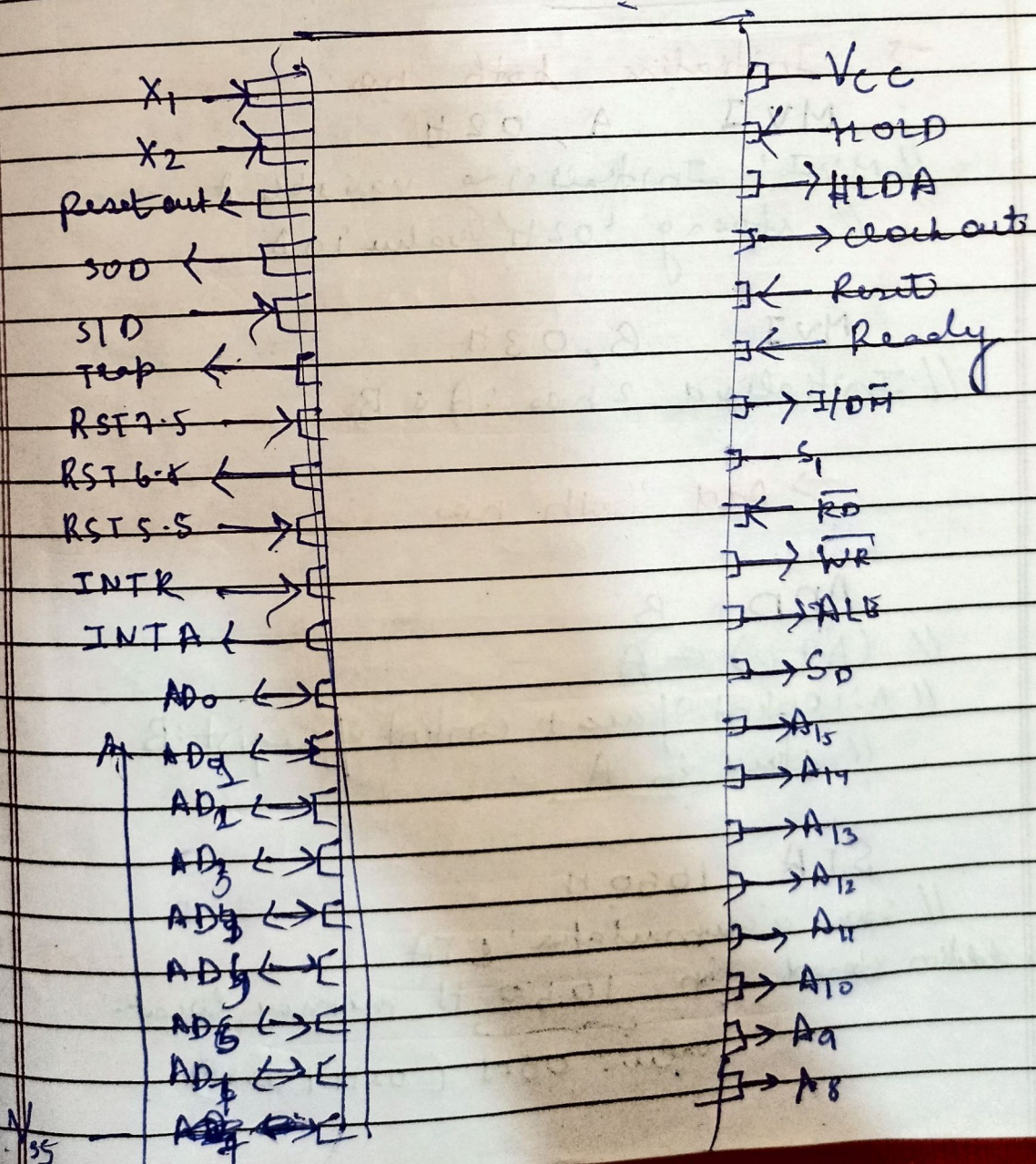
(5) ~~Address data bus/ buffer~~ : bidirection

(6) Power supply : single lines

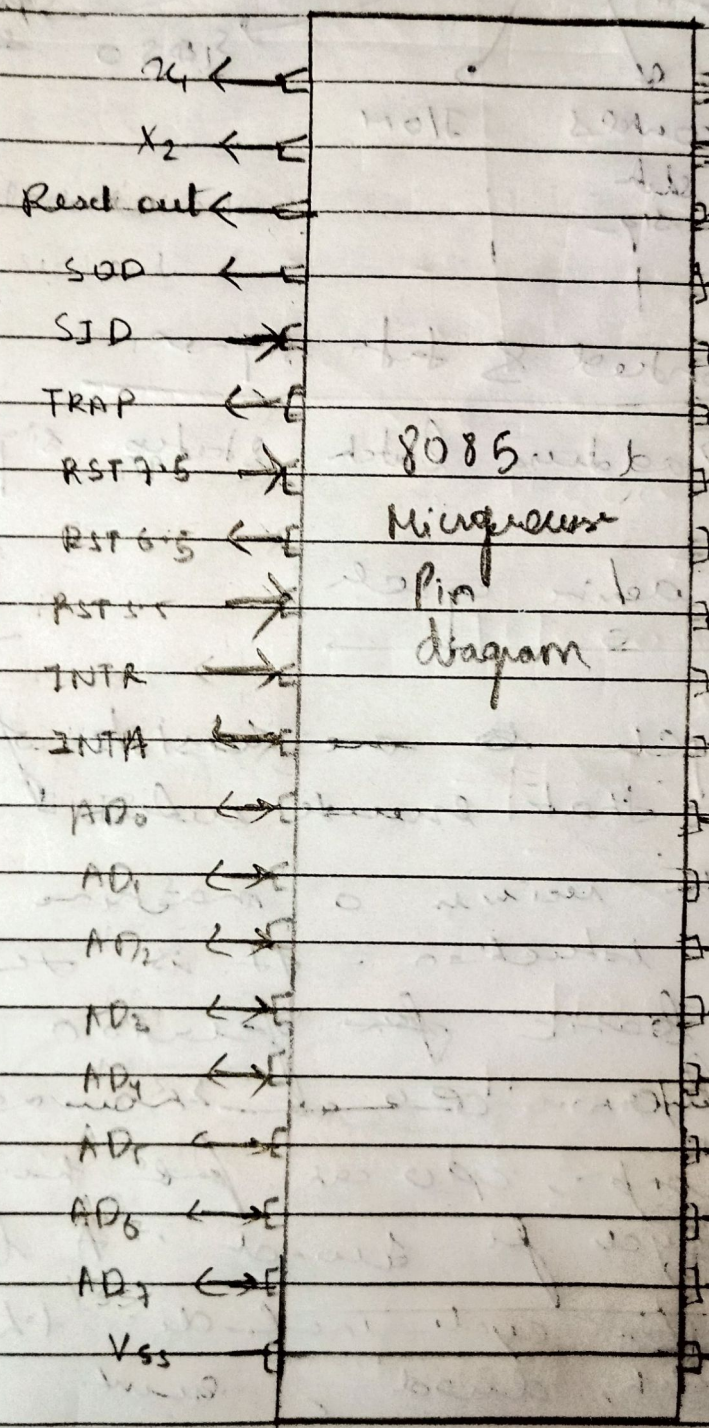
(7) DMA : ~~out~~ HOLD, in
HLDA, out

Page No.
Date :

(8) Clock → out
(9) Reset → In
(10) Ready → In
(11) status : out
(12) control : RD → In, WR → out, ALE → out
(13) status clear : out
(14) Address bus / buffer : out



Page No.
Date :


8085
Microprocessor
Pin
diagram
X1
X2
Reset out
SOD
SID
TRAP
RST 7.5
RST 6.5
RST 5.5
INTR
INTA
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
Vss
Vcc
HOLD
HLDA
CLK-OUT
Reset in
Ready
IO/M
S1
RD
WR
ALE
S0
A15
A14
A13
A12
A11
A10
A9
A8

Instruction set

CLASSIFICATION OF INSTRUCTIONS

- **Data transfer**
  transfer data
  R → R
  M → R
  R → M
- **logic grp.**
  logical operations (OLD AND, OR etc)
- **Stack I/O & Machine control Inst.**
  control Stack, I/O, Machine hardware (PUSH B, etc)
- **Branch grp**
  change sequence of flow of program. (Ex: JMP, JNZ)
- **Arithmetic group**
  arithmetic op.

0/0/2022
~~9/10/22~~

Evergreen
Page No.
Date : / /20

UNIT - 1

8085 : Instruction set

1] helps in programing ← { Data transfer group

divided in 6 groups.

data goes from one place to another.

① MVI R, data
used to Initialise a register.
to store a value in a register.

Eg: MVI B, 25H

I: Immediate

There is data inside the register only.

B ← 25H

② LXI Rp ; Data
Load Immediately

X: register pair

I: data inside & Instruction.

Eg: LXI B, 2000H
↓ B-C pair. ↳ data.

BC ← 2000H
[B ← 20H
C ← 00H]

Page No.
Date : / /20

3) MVI M, data

Eg: MVI M, 25H

↓

Memory location where address pair is given by H-L pair

4P

H	L
40	00

[HL → 4000]

↓

HL Pair has memory location of 4000.

M ← 25H

[HL] ← 25H

↳ 25H to memory location pointed by HL.

~~HL~~ a memory is stored in HL. in that memory we are storing data.

4000 | m

location 4000 → M

4) MOV Rd, Rs → same register

Eg: MOV B, C

B ← C

25

↓

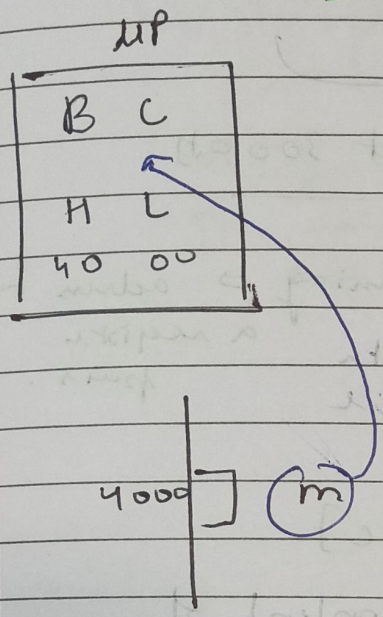
Destination register

*: value of C copied in C. The value is never cut. It always copied is.

5) MOV B, M

B ← M

Memory location pointed by H-L pair.



6) MOV m, C

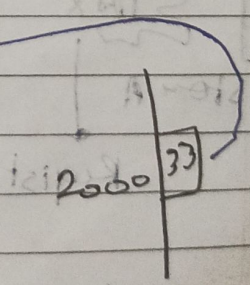
m ← C

value of C to memory location pointed by H-L pair.

7) LDA 2000H // direct addressing ; fetching 8 bit data from memory

load A

A ← [2000H]



NO I in this Instruction. Thus, there is no data in the Instruction. so, 2000H is the address from whe content is going to be stored in A.

8] STA 3000H.

store A

A gives value.

A give value.
so, store value by A.

[3000H] ← A

store A at 3000H

9] LDAX B // Indirect addressing ⇒ address for a register pair.

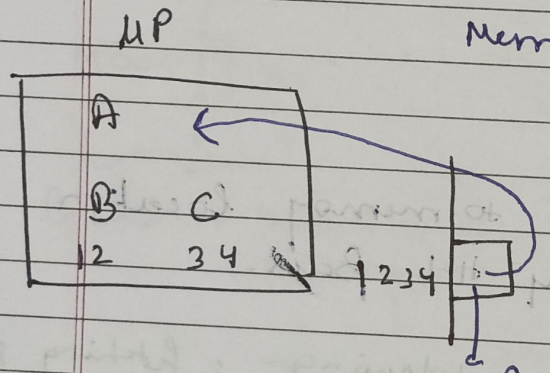
load A

That means register pair

Register Pair.

A ← [BC]

A gets the contents of Memory location pointed by B-C pair



value stored at 1234.

10] STAX D // Indirect addressing mode

store A

Register Pair.

[DE] ← A

11] LHLD 2000H // fetching 16 bit data from mem.

load HL

direct addressing

HL ← [2000H]

{L ← [2000H]
H ← [2001H]}

lower byte, lower address
higher byte, higher address

12] ~~PCHL~~ SHLD 3000H

store ~~in~~ H-L pair

Direct addressing.

H-L pair will give value, store that value/content.

[3000H] ← [H-L]

{ [3000H] ← L
[3000+H] ← H }

∴ RARELY USED INST :

13] PCHL

PC ← HL

contains address of next instruction.

14] SPHL

SP ← HL

allocates stack.

Stack pointer.

↳ address of top of stack.

15] XCHG

Exchange

HL ⟺ DE

WZ register used as helper.

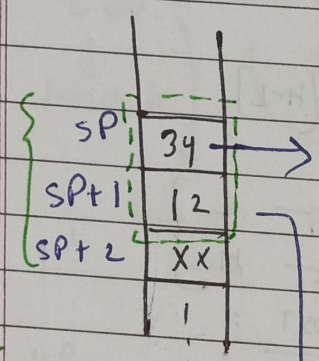
→ rarely used Instruction; Imp for Theory.

Evergreen
Page No.
Date : / /20

16] XTHL

Exchange top with [H-L] } we have to exchange the data present at the top of the stack with The data present in [H-L].

top of stack



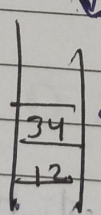
→ top of stack.

address of data present in stack

after implementing XTHL; The data will exchang with 34,

SP will have new data.

since, [H-L] pair have 16 bit data, SP & [SP+1] both will change.



Exchange ↔ with H

↔ Exchange with L (lower address - L pair)

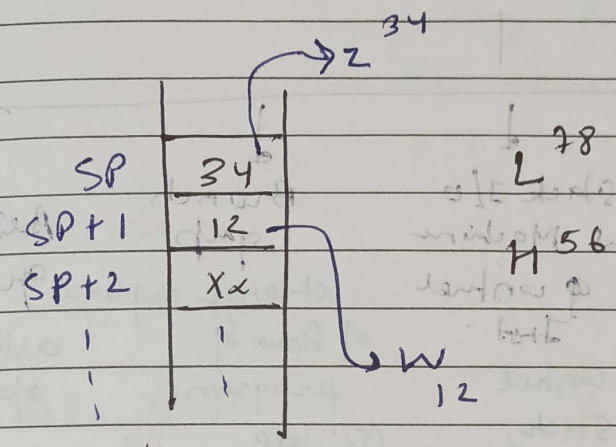
(Higher address → high bit → H)

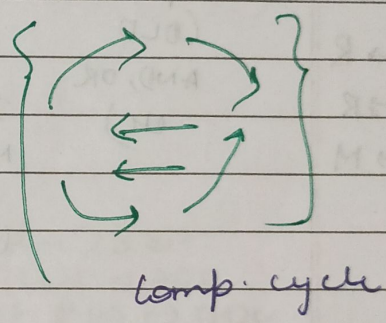
* we have to EXCHANGE [H-L] with [S] & [S+1].

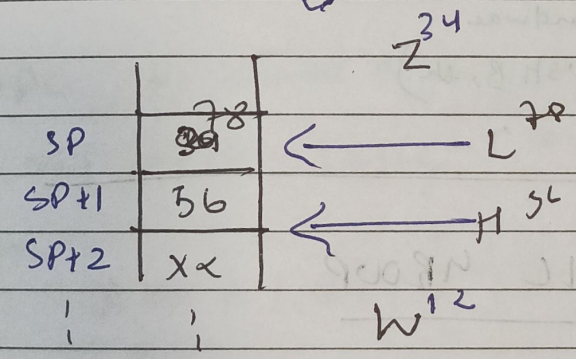
we do not wish to loose the value present at [S] and [S+1]. we wish to just

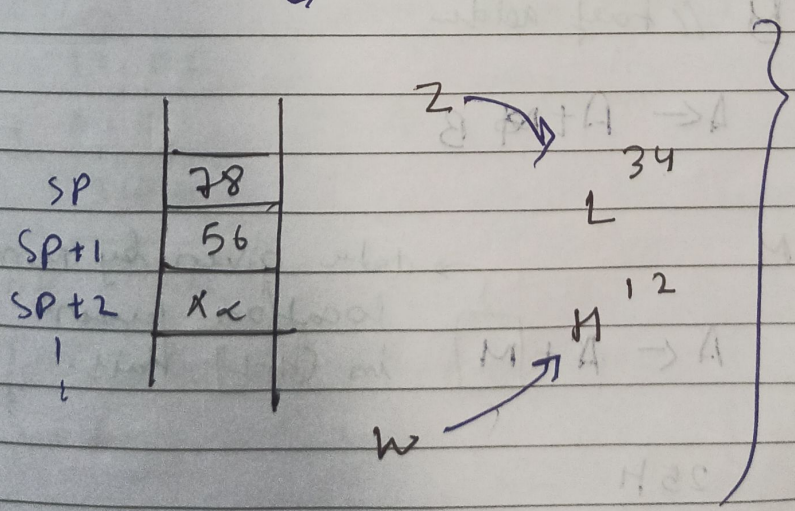
Change the value without loosing it.

How does this takes place?









ARITHMETIC GROUP

1) ADD B // half adder

$A \leftarrow A + B$

2) ADD M

$A \leftarrow A + \boxed{M}$ → data given by Memory location pointed in [H-L] Pair.

3) ADI 25H

$A \leftarrow A + 25$

4] SUB B

$A \leftarrow A - B$

↳ always holds the first operand.

5] SUB M

$A \leftarrow A - M$

6] SUI 25H

$A \leftarrow A - 25H$

7] ADC B → used to carry number > 8 bit
↳ lowest 8 bit → ADD
rest Bits → ADC

add with carry

full adder

$A \leftarrow A + B + CF$ } carry from previous addition.

↳ carry flag (0/1)

Eg:

```
CF
  ①  1
  12 FF
+ 00 01
-------
  13 00
-------
HB   LB
```

} 16 bit addition

↳ to add then → ADD.
↳ then, used ADC

} Two instruction used to add 16 bit no. in 8085.

8) ADC M

$A \leftarrow A + M + CF$

9) ACI 25H

$A \leftarrow A + 25H + CF$

10) SBB B

subtract w/ borrow

$A \leftarrow A - B - CF$

$$\begin{array}{r} 1\,\cancel{2}\,\cancel{1}\,7 \\ -\ 1\,9 \\ \hline 0\,8 \end{array}$$

same concept as ADC

11) ~~SB~~ SBB M

$A \leftarrow A - M - CF$

12) SBI 25H

$A \leftarrow A - 25H - CF$

13) INR B // affects flags.

$B \leftarrow B + 1$

If we ~~increment FF~~ → ; it will become 0

Page No.
Date : / /20

14) INX B // does not affect flags, since, it increment 16 bit; so, it affects Incremental / decremental.

BC ← BC + 1

16) INR M

M ← M + 1

16) DCR B

B ← B − 1

17) ~~DCR~~ DCX B

BC ← BC − 1

18) DCR M

M ← M − 1

19) DAD D → adds 16 bit number.
double addition.

requires to 4 status DE - pair.

* In DAD; ~~accum~~ (H-L) pair acts as accumulator of this instruction.

HL ← HL + DE

* If : DAD H
↳ HL ← HL + HL } 16 bit no. multiplied by 2.

2] **DAA** Decimal adjust after addition

A add

high nibble

if HN > 9

or

CF = 1

then add 60

lower nibble.

if LN > 9

or

AC = 1 (auxiliary carry)

then,

add 6

Eg: 24 H
25 H
49 H

HN LN

since, LN ≤ 9 &
HN < 9; no adjustment hai

Eg: 25 H
25 H
4A H

25 : 0010 0101
+ 25 : 0010 0101
: 01 0010 10

4 10
4 A

LN > 9 → LN + 6
4A + 6
50 } in decimal

4A
4B
4C
4D
4E
4F
50
} 4A + 6

Output

Eg: 28 H
28 H
---
50 → wrong ans ☹ ∴
56 here comes the CARRY (CF)

since, AC = 1 ; add 6 !

Eg : 99 H
99 H

Ideal ans : 198 H

AC
1
99 H
99 H
---
1 32 H hexadecimal addition
CF

DAA on

~~132~~
1
99 H
99 H
---
1 32 H

; CF = 1, 9 > HN LN < 9 ; ~~CF~~ AC = 1
9 3 = 6 + 3 2 + 6 → 8

19 8 H

3) Logic group

(a) ANA R

Eg: ANA B

$A \leftarrow A \wedge B$

AND	
0 0	0
0 1	0
1 0	0
1 1	1

(b) ANA M

$A \leftarrow A \wedge M$

Clear NB:

A =	0011	0101 235
ANI, OFH	1111	0000
	0011	0000

HN as it is ($\wedge$ w/ 1)

cleared the LN by $\wedge$ them w/0

(c) ANI 25H

$A \leftarrow A \wedge 25H$

Date : / /20

Set LN

OR	
00	0
01	1
10	1
11	1

ORI, 0FH

A =	0011	0101
	0000	1111
	0011	1111

(d) ORA R

A ← A ∨ R

(e) ORA M A ← A ∨ M

(f) ORI 25H A ← A ∨ I

~~XRO~~

XOR

XOR	
00	0
01	1
10	1
11	0

(g) XRA R

A ← A ⊕ R

(h) XRA M A ← A ⊕ M

(i) XRI 25H A ← A ⊕ I

complement LN

XRI, 0FH

A =	0011	0101
	0000	1111
	0011	1010

(j) CMP B

A − B cf

	~~AF~~ CF	ZF
A > B	0	0
A = B	0	1
A < B	1	0

(k) CMP m

A − m

(l) CPI 25H

A − 25H

(m) STC

CF = 1

(n) CMC

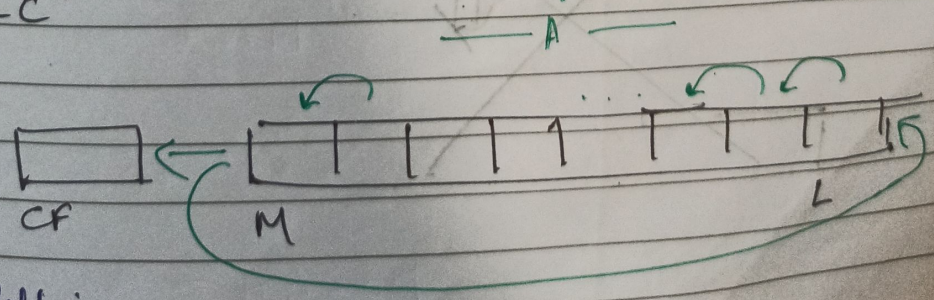
CF = $\overline{CF}$

(o) CMA

A = $\overline{A}$

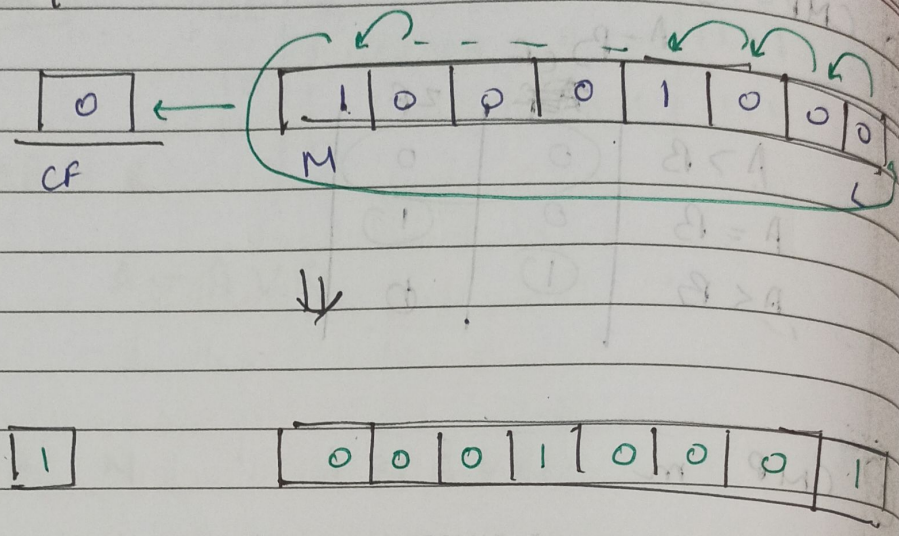
(p) RLC:

every bit rotating to their left.



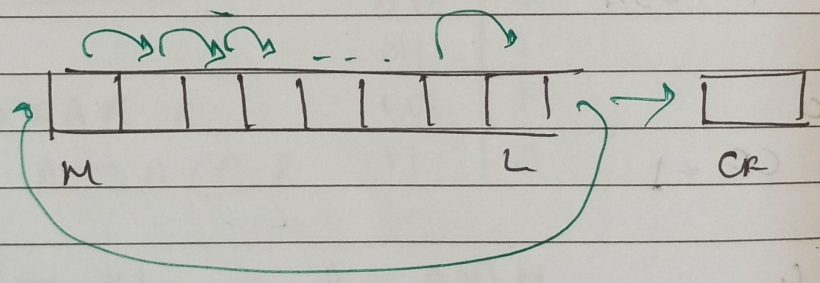
Rotate left: MSB will go to CF & LSB

Eg:

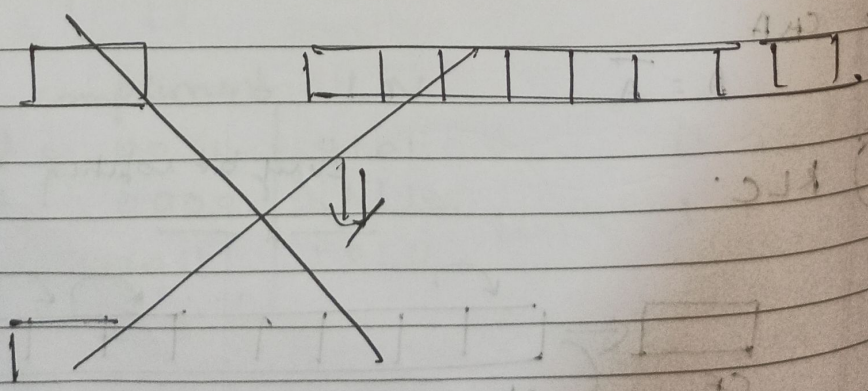


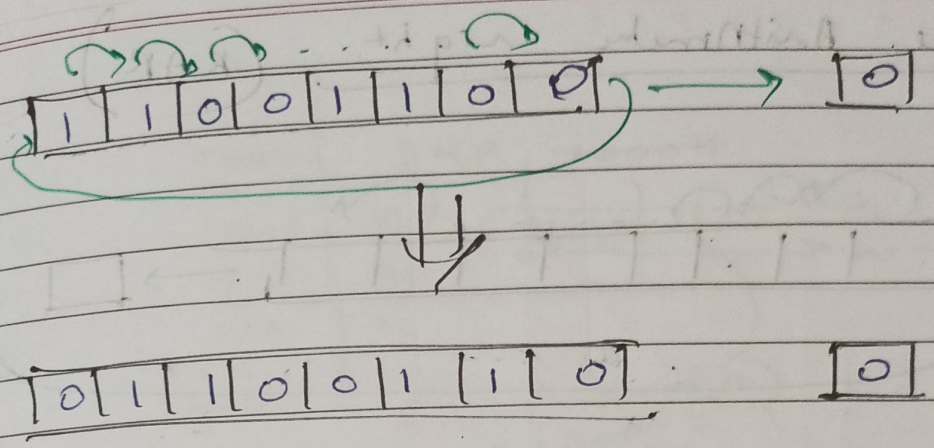
(q) RRC

Rotation towards right.

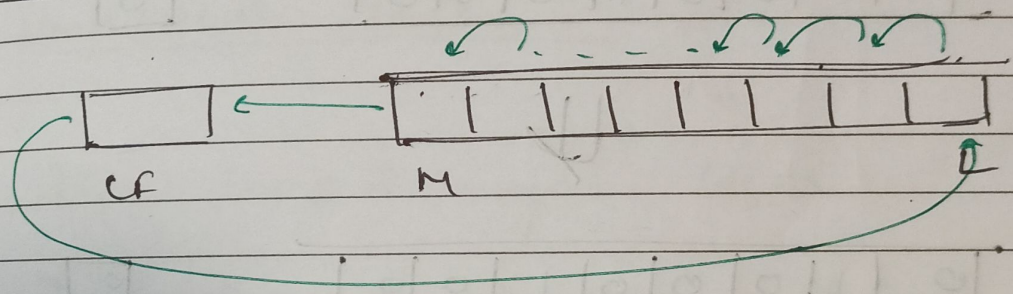


Eg:

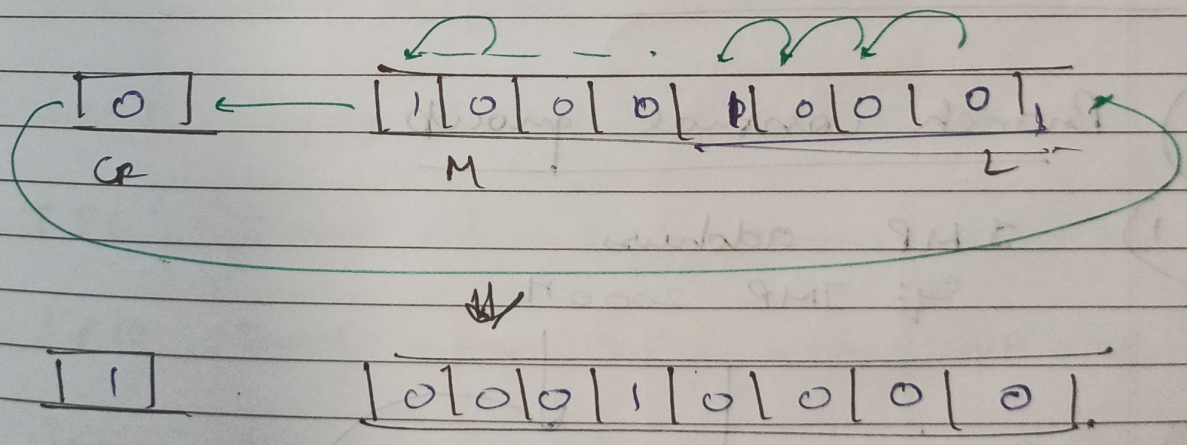




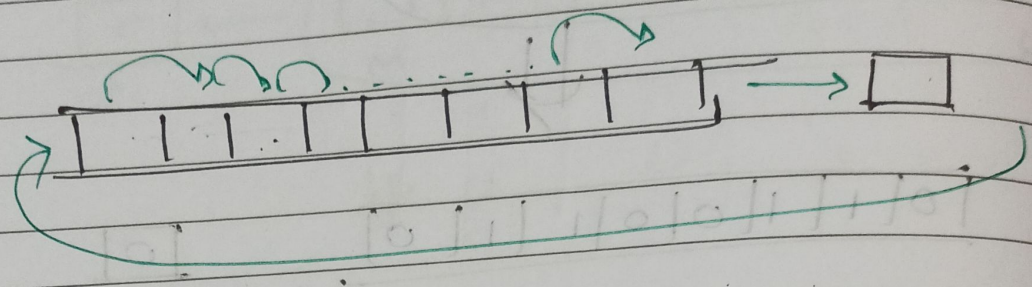
(2) RAL - Rotate arithmetic left (along with CF)



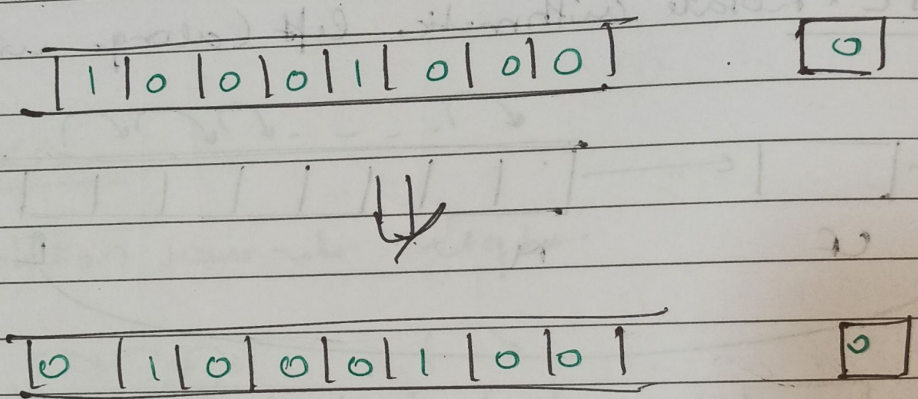
Eg:



(4) Rotate Arithmetic right (RAR)



Eg:

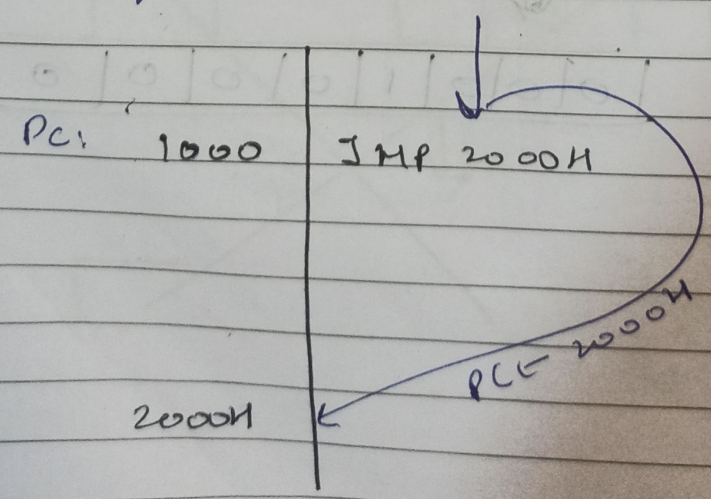


---

4) Branch control group

1) JMP address

Eg: JMP 2000H



PC:	1000	JMP ~~20004~~
PC:	1001	00 → Z
PC:	1002	20 → W
PC:	1003	X
	2000	

PC ← 2000 (W Z)

2) <u>conditional JMPs</u>

JC Jump if carry CF = 1

JNC Jump if no carry CF = 0

JZ Jump if zero ZF = 1

JNZ Jump if not zero ZF = 0

JPE Jump when Parity even PF = 1

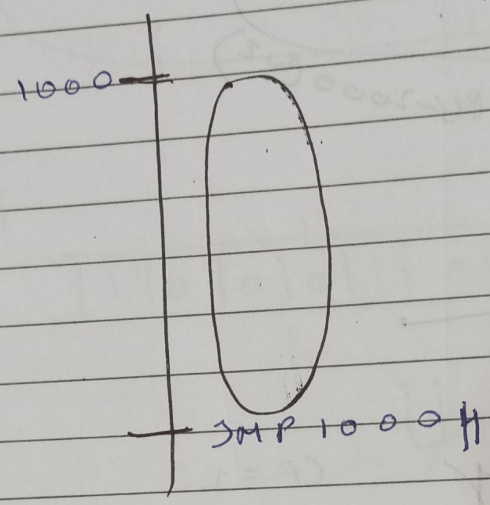
JPO Jump when Parity odd PF = 0

JM Jump when minus SF = 1

JP Jump when plus SF = 0

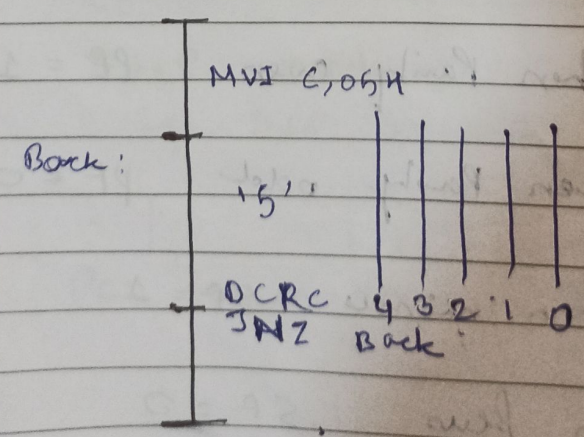
3) Methods of looping

∞ loop



→ ∞ loop used in clocks

finite loop



Back: MVI B, 08H

8

7 6 5 4 3 2 1 0

DCR B

JNZ Back

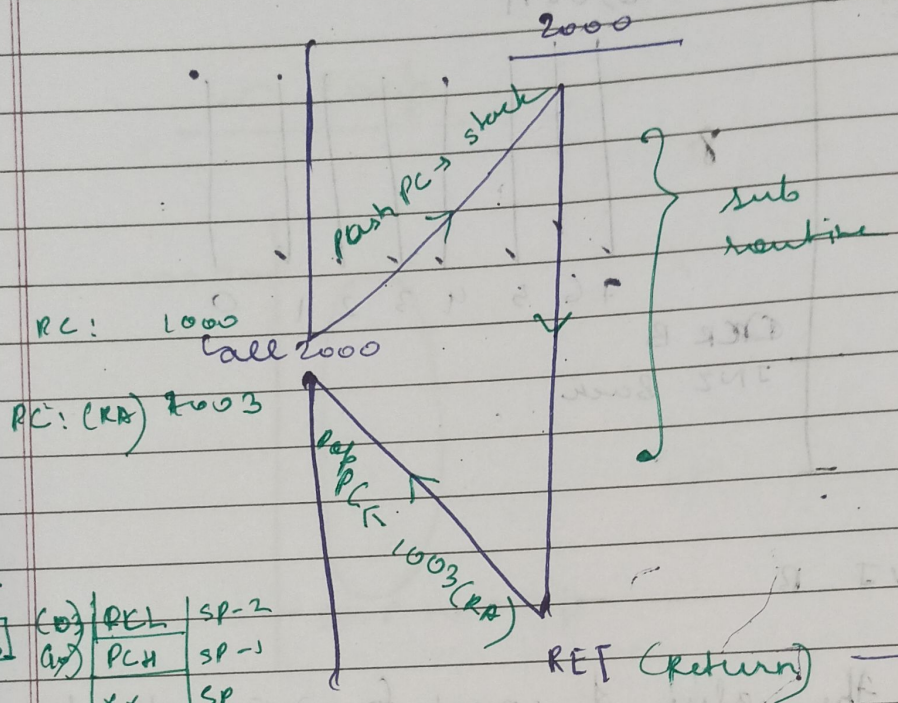
Q) MVI B _____

tell the value of count for 256 iterations

options: 00, 1, ff, not possible

ans → 0

4) Call



PC

10	03

(03)	PCL	SP-2
(04)	PCH	SP-1
	XX	SP

10 03

PCH	PCL

PC

RET (Return)

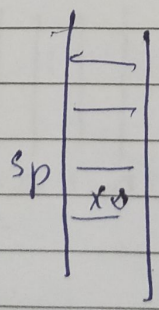
This instruction is written at the end of the ~~main routine~~ sub routine & enables the control to go back to the main program.

call 9000H

1) Push PC → stack

PC

PCH	PCL
30	03



2) PC ← 9000 H

Evergreen
Page No.
Date : / /20

5) $RST_n \approx$ call $(n \times 8)$ { software interrupts of 8085 }
$(n = 0 \ldots 7)$

RST 0 – 0000
RST 1 – 0008
RST 2 – 0010
RST 3 – 0018
RST 4 – 0020
RST 5 – 0028
RST 6 – 0030
RST 7 – 0038

8 increment

Standard functions are stored in these addresses.

19/9/2022

8085 Instruction Set

Stack I/O and Machine control group

Stack Instructions

1) PUSH Rp → only works on register addressing Mode.

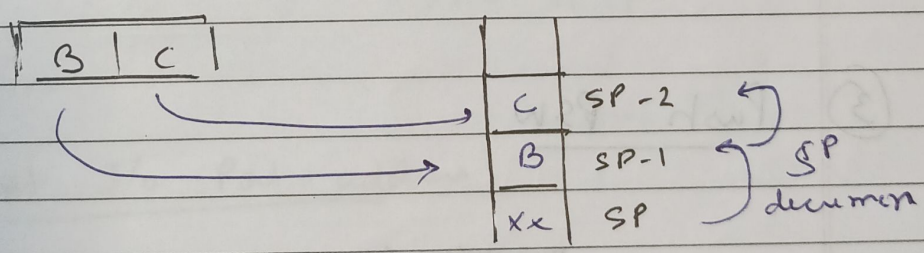
push the given register into the stack.

Eg: PUSH B

↓

(BC Pair)

16 bit ~~process~~ ~~first~~ operation



SP ← SP-1
[SP] ← B
SP ← SP-1
[SP] ← C

(2) POP R_p (16 bit)

Pops top two elements from the stack into the given register pair.

The lower byte comes out first, as the higher byte was pushed in first and stack operates in LIFO manner.

Ex: POP B ; C ← [SP]
SP ← SP+1
B ← [SP]
SP ← SP+1

data pop
and transfer to BC Pair

(3) Push PSW

| A | F |

It pushes PSW (program status word), into the stack.

The PSW is the combination of accumulator and the flag register, accumulator being higher byte

| A | F |

* PSW can only be used in PUSH and POP instructions.

Eg: PUSH PSW ; SP ← SP − 1
[SP] ← A
SP ← SP − 1
[SP] ← F

(4) POP PSW :

It pops the top two elements from the stack into the PSW.

Eg: POP PSW ; F ← [SP]
SP ← SP + 1
A ← [SP]
SP ← SP + 1

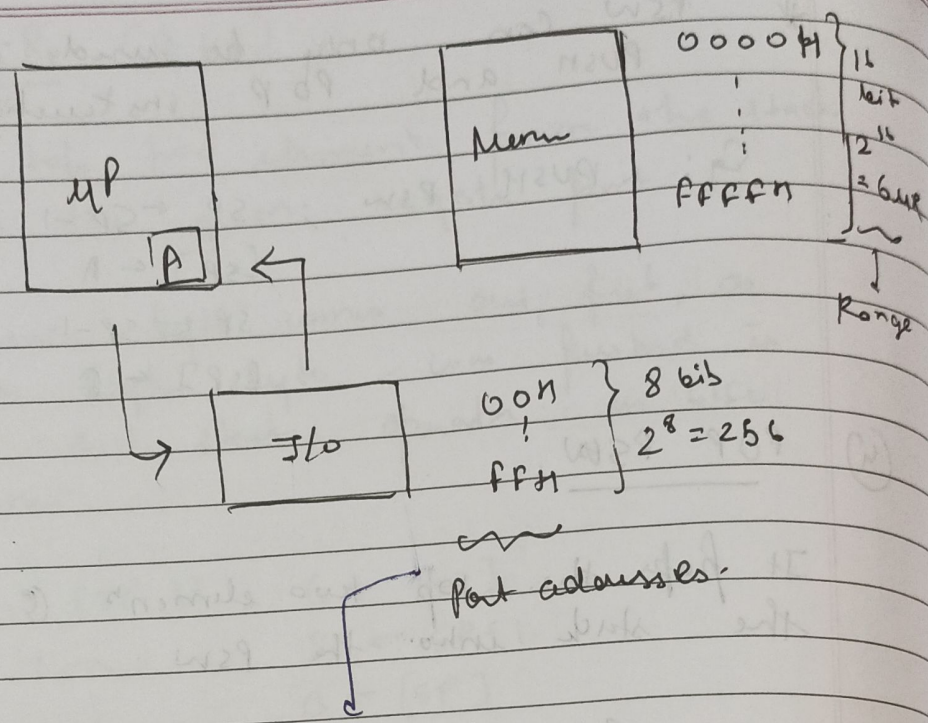
(5) In 8-bit I/O Port address

8085 has 256 I/O ports having 8 bit addresses 00H --- FFH.

This instruction is used to read data from an I/O Port, whose address is given in the instruction.

This data can be read into the accumulator only.

Eg: IN 80 [A ← [80]$_{I/O}$]



Small addr
because this enables us
to connect 256 devices
with a computer.

(6) OUT 8-bit I/O Port address

This instruction is used to send data from the accumulator to an I/O ~~port~~ port, whose address is in the instruction.

The data can be sent from Accumulator ONLY.

Eg: OUT 80; $[80]_{I/O} \leftarrow A$

(7) SIM (set interrupt Mask)

This instruction is used to set the Interrupt pattern for the μP.

The appropriate bit pattern is loaded into the accumulator and then this instruction is executed.

It is basically used to mask / unmask the interrupts except TRAP and INTR.

It can also be used to send a 'bit' out through the serial ~~pin~~ out pin SID.

(Mask → disable)

Eg:

SID : μP accepts the masking pattern through the accumulator

(8) RIM (Read Interrupt Mask)

Used to read the interrupt masking pattern for the μP.

~~After~~ After executing the instruction, the μP loads the bit pattern into ~~to~~ the accumulator.

It can also be used to recieve a 'bit' out through the serial in pin SID.

Eg:

RIM ; μP loads the masking pattern into the accumulator

(9) EI (Enable Interrupt)

used to enable the interrupts in μP.

This instruction sets ~~to~~ the INTE flip - flop (Interrupt enable flip flop).

This instruction effects all the instruction except TRAP.

Eg: EI ; $INTE_{F/F} \leftarrow 1$

(10) Disable Interrupt

used to disable Interrupts in µP.

This instruction resets the INTE flip flop.

This instruction effects all the interrupts except TRAP.

Eg: DI ; INTE ff ← 0

By default, Interrupts are disabled.
(on reset)

when the system is start working properly, it enables the disable

(11) NOP

No operation; but consumes time of µP.

This is the ~~way~~ simplest method of causing a software delay.

Eg: NOP ; - - - -

12) HLT (Halt)

This instruction signifies the end of the program

It causes the µP to stop fetching any further instructions, hence program ~~executed~~ execution is stopped.

It makes the Halt flip-flop inside the µP = 1.

µP checks the Halt flip flop in the beginning of the next machine cycle and stops all operations.

Eg:
HLT ; Halt flip flop ← 1.

8085

SIM & RIM

→ Multipurpose intrpt. (can mask intrpt, reset 7.5, send serial data)

I] SIM : Set Mask interrupt

work together...

Serial output data. ←

SOD	SDE	X	R7.5	MSE	M7.5	M6.5	M5.5

SOD: 1: send '1' on SOD
0: send '0' on SOD.

SDE: 1: send data on SOD
0: nothing

R7.5: 1: Reset RST 7.5
0: Nothing

MSE: 1: do masking
0: nothing

M7.5, M6.5, M5.5: 1: mask
0: unmask

II] RIM : Read mask interrupt

Serial input data ←

SID	I7.5	I6.5	I5.5	IE	M7.5	M6.5	M5.5

SID: 1: '1' present on SID
0: '0' present on SID.

I7.5, I6.5, I5.5: 1: pending interrupt
0: Not.

IE: 1: Interrupts are enabled
0: Interrupts are disabled.

M7.5, M6.5, M5.5: 1: masked
0: unmasked

Addressing modes

12/9/22

Addressing Modes

The manner in which an operand is given in an instruction.

operand → value on which we have to work on

1] Immediate addressing Mode (2/3 byte)

data is given in the instruction.

Eg: MVI B, 25 H

LXI B, 2000H } load immediately into BC pair (value 2000)

BC ← 2000

X means register pair.

Advantage:

→ Easy to load.

Drawback:

Size of Instruction becomes large.

* How to determine size of an Instruction?

→ An instruction can be of 1, 2 or 3 bytes.

→ opcode → 1 Byte
operand → 1 byte

MVI B, 25H → 2 bytes
opcode operand

MOV B, C → 1 Byte.
(opcode)

LXI B, 2000H → 3 Byte
opcode 1 B — operand 2 Byte

Push B → 1 Byte — OUT 60H → 2 byte

LDA 2000H → 3 byte — MOV B, C → 1 Byte

STA 3000H → 3 byte — STC → 1 Byte

IN 20H → 2 byte — ADI 25H → ~~1~~ 2 Byte

SUI 6AH → 2 byte — SHLD 2000H → 3 Byte.

II] Register addressing mode (always 1 byte)

data is given to a register
Eg: MOV B, C ; B ← C (value of C copied to B)
INR B ; B ← B+1
INX B ; BC ← BC+1

Disadvantage
→ instructions a bit complex

Advantage
→ size of Instruction is always 1 byte.
If size of Instructions / program becomes smaller, then the program becomes more efficient and faster.

* Example of a program

Initialize : A=0, B=0, C=0

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

OR

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

III] Direct addressing mode (always 3 bytes)

address is given in an instruction

Eg: LDA 2000H A ← [2000H]
STA 3000H [3000H] ← A

→ advantage
Simple

→ drawback
always makes 3 bytes instruction

* Ex: Add two num.

```
LDA 2000H
MOV B, A
LDA 3000H
ADD B
STA 4000H
HLT
```

IV] Indirect addressing Mode

address given in a register.

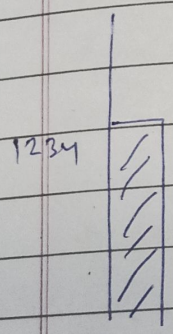
Ex: LDAX B ; A ← [BC]

~~value of address at [BC] to AC.~~

A will get contents of memory location pointed by BC ~~&~~ B-C.

STAX D ; A [DE] ← A

Ex: get me the data of 100 locations starting from 1234.



LXI B, 1234H
LDAX B, } In loop
INX B

advantages of indirect addressing mode

→ address is given in a register which we can increment or decrement to access a series of location.

Eg: Store data at 2000 to 3000 by A

LDA 2000H
STA 3000H
HLT

Q) write a program to copy a block of data from 2000 to 3000

~~LDAX B,C~~
~~LDA~~ LDX D, 3000H
LDX B, 2000H
loop { LDAX B
~~INX B~~ STAX D
INX B

LXI B, 2000H BC ← 2000H
LXI D, 3000H DE ← 3000H
loop { LDAX B
STAX D
INX B
INX D

V) Implied addressing mode

operand is 'implied'
Eg: STC set the carry flag: CF ← 1
CMC CF ← $\overline{CF}$

Programming questions

Q) ~~2 8 add~~

MVI A, 02H
// MVI: Initialising a variable A
// Storing A

add 2 ~~bit~~ nos. 8 bit each

→ Initialise both nos.

MVI A, 02H
// MVI: Initialising variable A
// storing '02H' value in A

MVI B, 03H
// Initialised 2 nos: A & B

→ Add both nos

ADD B
// (A+B) ← A
// A: content of acc + content of regist: B
// stored in A

STA 1050H
// store accumulator: STA
addition stored in 1050H memory locat.
value: 05H (02H+03H)

Page No.
Date :

HLT
// stop the execution of program.

add 2 No; fetch them from memory location and add them and store them.

05H	1000H
07H	1001H
	1002H

} Fetch this data, add & store in 1002H.

→ Input data

LXI H, 1000H
fetches Input present in memory 1000H

MOV A, M
Content of memory moved to A (05H)
M ← A ← 05H

→ Input 2nd data

INX H
Increment the memory pointer
(H ← 1001H)
and, data stored in Memory
A ← 07H

→ Add

ADD M
A ← A + M

Date :

STA 1002 H

HLT

07H + 05H → 0CH → Hexadecimal

12H

Addition of two 8-bit nos. w/ carry.

F1 → 11110001

15 → 00010101

1 00000110

carry

8 Bit no.

Algo: Inp (1st no.)

↓

Inp (2nd no.)

↓

Add

↓

no carry → store result

carry → store carry → store result

↓

End

Page No.
Date :

→ Input 1st no.

LXI H, C050 H
C050H ← F1H

MOV A, M A ← F1H

→ Input 2nd no

INX H
B C051H ← 15H

→ Initialize a register for carry for

MVI B, 00H

→ Add

ADD M
A ← F1H + 15H ← 06H

JNC loop
conditional jump statement : jump if carry not available. / Jump if no carry.

Jump to loop.
INR B // B ← B+1 ← 01H
loop STA C053H
MOV A, B
A ← B
STA C052H
HLT

Date :

Addition of 2 16-Bit no.

1st → 0402H
2nd → 0607H
0A09H

HL: for 16 bit no.
H: for 8 bit no.

1st → A645H
2nd → 9B23H
[1] 4168H
carry

H L
↓ ↓
MSB LSB

```
LHLD C050H
XCHG
// data of HL register
   now in DE register
LHLD C052H
```

C050H	45H
C051H	A6
C052H	23
C053H	9B
C054H	
C05FH	

D E H L
A6 45 9B 23

```
DAD D
MVI C,00H
JNC loop
INR C
loop SHLD C054 →
MOV A,C
STA C056H
HLT
```

C054H	C055H
68	41

Date :

1's complement : 8 bit no.

$A \rightarrow$ 10101110

$\bar{A} \rightarrow$ 01010001

```
LDA 6050H
CMA
STA 6051H
HLT
```

2's complement : 8 bit no.

$A \rightarrow$ 10101110

1's comp. $\bar{A} \rightarrow$ 01010001

$+ \quad 1$

2's comp $\rightarrow$ 01010010

```
LDA 6050H
CMA
~~STA~~ INR A
~~STA 6051H~~
HLT
```

Page No.
Date :

1's comp. : 16 bit no.

~~LHLD 4050H~~
~~CMA~~
~~STA 4051H~~

LXI H, 4050H

// data stored in 4050H: LSB stored in H

MOV A, M
CMA
STA 4052H

INX H
MOV A
CMA
STA 4053H

HLT

Date :

2's complement of 16-Bit no.

Subtraction of two 8-bit no

One way

```
MVI A, 05H
MVI B, 03H
SUB B
STA C000H
HLT
```

~~MVI H, C050H~~
~~MVI MOV A, M~~

```
LXI H, C050H
MOV A, M
INX H
SUB M
STA C052 H
HLT
```

BCD ~~subtraction of two no~~

~~(8 bit decimal subtraction)~~

Page No.
Date :

Div. of two 8-bit nos.

10 ÷ 2 → repeated sub

divider divs.

```
  10
 - 2
 ----
  08
 - 2
 ----
  06
 - 2
 ----
  04
 - 2
 ----
  02
 - 2
 ----
   0
```

2 → 5

QC = 5

RC = 00

```
LXI H, C050H
MOV B, M
INX H
MOV C, M
MVI D, 00H
MOV A, B
loop SUB C
INR D
CMP C
JNC loop
STA C053H
MOV A, D
STA C052H
HLT
```

Page No.
Date :

Multiplication of two 8-bit no.

64H × 03H = 124H

04 + 04 + 04 = 124

```
LXI H, 8050H
MOV B, M
MVI A, 00H
MVI D, 00H
INX H
MOV C, M
loop2 ADD B
      JNC loop
      INR D
loop  DCR C
      JNZ loop2
      STA 8052H
      MOV A, D
      STA 8053H
      HLT
```

Programming - 8085

Q) Add 2-8 bit numbers stored at 2000H and 2001H.

Store sum at 2002H and carry at 2003H.

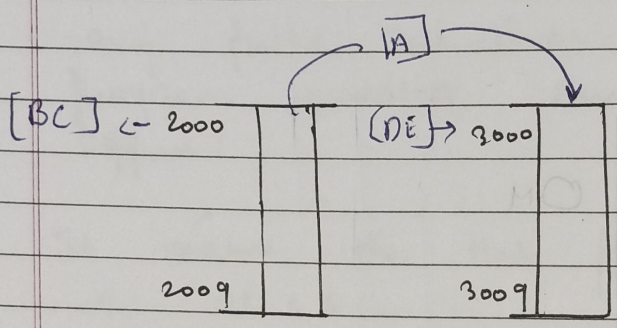
(use M)

HL = 2000	0M
2001	
2002	
2003.	

2999H	MVI C, 00H	(C ← 00H)
3000H	LXI H 2000H	(HL ← 2000; M ← [2000])
3003H	MOV A, M	(A ← M)
3004H	INX H	(M ← [2001])
3005H	ADD M	(A ← A + M)
3006H	JNC 3010H	(Jump if no carry)
3009H	INR C.	(C ← C+1)
3010H	INX H	(M ← [2002])
3011H	MOV M, A	(M ← A)
3012H	INX H	(M ← [2003])
3013H	MOV M, C	(M ← C)
3014H	HLT	(HLT F/F ← 1)

Q) Block transfer Program

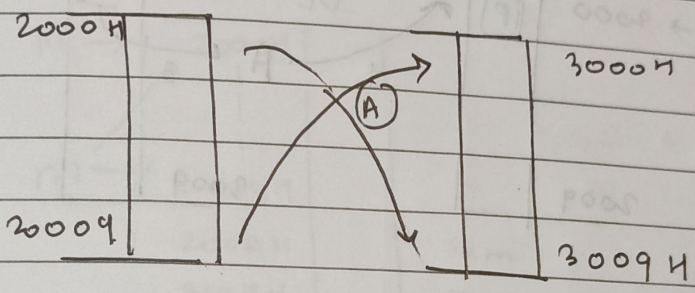
Transfer a block of 10 bytes from ~~100~~ 2000H to 3000H.



4000H	LXI	B, 2000H	([B-C] ← [2000])
4003H	LXI	D, 3000H	([D-E] ← [3000H])
4006H	MVI	L, 0AH	(L ← 0AH)
4008H	LDAX	B	(A ← [B-C])
4009H	STAX	D	([D-E] ← A)
4010H	INX	B	([B-C] ← [B-C]+1)
4011H	INX	D	([D-C] ← [D-C]+1)
4012H	DCR	L	(L ← L-1)
4013H	JNZ	4008H	(Jump if not zero)
4016H	HLT		(HLT$_{F/F}$ ← 1)

Q) Inverted block transfer program

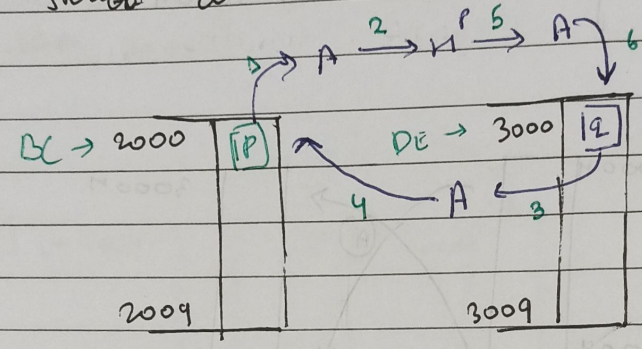
Perform inverted block transfer of 10 bytes from 2000H to 3000H.



4000H	LXI B 2000H		(BC ← 2000H)
4003H	LXI D 3009H		(DE ← 3009H)
4006H	MVI L 0AH		(L ← 0AH)
4008H	LDAX B		(A ← [B-C])
4009H	STAX D		([D-E] ← A)
4010H	INX B		(BC ← BC+1)
4011H	DCX D		(DE ← DE-1)
4012H	DCR L		(L ← L-1)
4013H	JNZ 4008H		(Jump if not zero)
4016H.	HLT		(HLT F/F ← 1)

Q) Block exchange Program

Exchange two blocks of 10 bytes stored at 2000H and 3000H.

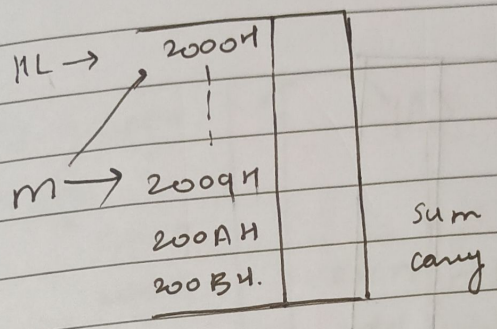


4000H	LXI	B	2000H	([BC] ← 2000H)
4003H	LXI	D	3000H	(DE ← 3000H)
	~~LA~~			
4006H	MVI	L	0AH	(L ← 0AH)
4008H	LDAX	B		(A ← [BC])
4009H	MOV	H, A		(H ← A)
400AH	LDAX	D		(A ← [DE])
400BH	STAX	B		([BC] ← A)
400CH	MOV	A, H		(A ← H)
400DH	STAX	D		([DE] ← A)
400EH	INX	B		(BC ← BC+1)
400FH	INX	D		(DE ← DE+1)
4010H	DCR	L		(L ← L-1)
4011H	JNZ	4008H		(Jump if not zero)
4014H	HLT			($HLT_{F/F}$ → 0)

Evergreen
Page No.
Date : / /20

Q) Add a series of numbers

Add a series of 10 numbers stored from 2000H.
Store result at 200AH and 200BH.

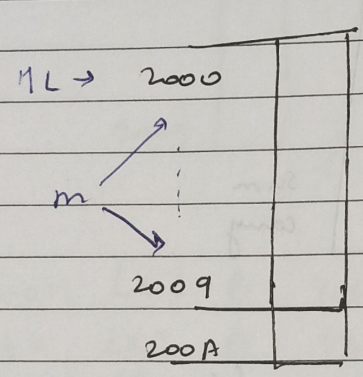


3000H	LXI H, 2000H	([HL] ← 2000H)
3003H	MVI A, 00H	(A ← 00H)
3005H	MVI B, 00H	(B ← 00H)
3007H	MVI C, 0AH	(C ← 0AH)
3009H	ADD M	(A ← A + M)
	~~INX H~~	
300AH	JNC 300EH	(Jump if no carry)
300BH	INC B	(B ← B + 1)
300EH	INX H	(HL ← HL + 1)
300FH	DCR C	(C ← C - 1)
3010H ~~300EH~~	JNZ 3009H	(Jump if not zero)
3013H	~~STA 200AH~~	
	~~INX H~~	
3014H	~~ST~~ MOV M, A	(M ← A)
3015H	INX H	(HL ← HL + 1)
3016H	MOV M, C	(M ← C)
3017H	HLT	(HLT F/F → 1)

Q) Find Highest number.

Find the highest ~~a~~ is a series of ten 8 bit numbers stored from ~~2000H~~ 2000H.

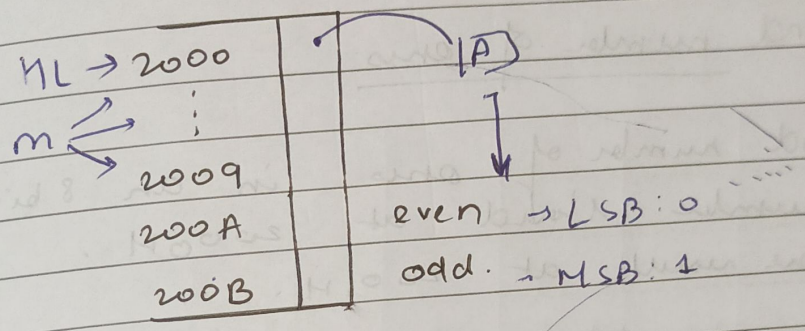
Store result at 200A H.

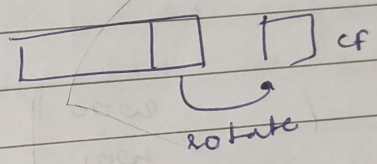


3000H	LXI H, 2000H	(HL ← 2000H)
3003H	MVI A, 00H	(A ← 00H)
3005H	MVI C, 0AH	(C ← 0AH)
3007H	CMP M	(A - M)
3008H	JNC 300CH	(Jump if no carry)
300BH	MOV A, M	(A ← M)
300CH	INX H	(HL ← HL + 1)
300DH	DCR C	(C ← C - 1)
300EH	JNZ 3007H	(Jump if not zero)
3011H	MOV M, A	(M ← A)
3012H	HLT	(HLT F/F → 1)

Q) Find even and odd numbers

Find the number of even and odd numbers in a series of 10 8 bit numbers stored from 2000H. Store result at 2000A H and 200B H.





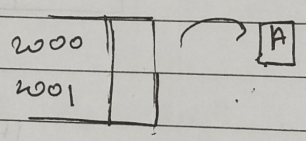
if CF → 1, odd
CF → 0, even

	~~Lxe~~	
3000H	LXI H, 2000H	(HL ← 2000H)
3003H	MVI E, 00H	(E ← 00H)
3005H	MVI B, 00H	(B ← 00H)
3007H	MVI C, 00H	(C ← 00H)
3009H	MOV A, M	(A ← M)
300AH	RRC	(Rotate right accumulator)
300BH	JNC 3012H	(Jump if no carry)
300EH	INR B	(B ← B+1)
300FH	JMP 3013H	(Jump to 3013H)
3012H	INR E	(B ← E+1)
3013H	INX H	(HL ← HL+1)

3014H	DCR C	(C ← C-1)
3015H	JNZ 3009H	(Jump if not zero)
3018H	MOV M, E	(M ← E)
3019H	INX H	(HL ← HL+1)
301AH	MOV M, B	(M ← B)
301BH	HLT	(HLT F/F → 1)

Q) <u>find number of ones</u>

find number of ones in an 8 bit number stored at 2000H. store result at 2001H.



3000H	LDA 2000H	(A ← [2000H])
3003H	MVI B, 00H	(B ← 00H)
3005H	MVI C, 08H	(C ← 08H)
3007H	RRC	(Rotate right accumulator)
3008H	JNC 300CH	(Jump if no carry)
300BH	INR B	(B ← B+1)
300CH	DEC C	(C ← C-1)
300DH	JNZ 3007H	(Jump if not zero)
3010H	MOV A, B	(A ← B)
3011H	STA 2001	([2001] ← A)
3014H	HLT	(HLT F/F → 1)

Architecture 8085

8085 architecture

Accumulator

It is an 8 bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal bus and ALU.

Arithmetic & logic unit

As the name suggests, it performs arithmetic & logical operations like addition, subtraction, AND, OR etc. on 8-bit data.

general purpose register

- There are 6 general purpose registers
- They are 8-bit general purpose registers
- They can also be used to store 16 bit data in register pairs.
- The possible register pairs are: B-C
  D-E
  H-L
- The HL pair also holds the address for the memory pointer M.

Temporary register pair (WZ, 16 bits)

- 16-bit register pair
- It is used by µP to hold temporary values in some instructions like CALL / JMP / LDA
- The programmer has no access to this register pair.

INR / DCR register (16 bits)

- It is used to Increment program counter after every instruction byte is fetched.
- It also increments or decrements stack Pointer after a Pop or push operation respectively.
- It is not available to the programmer.

Temporary register (8 bit)

- It is used by µP for storing one of the operands during an operation.
- The programmer has no access to this register.

FLAG REGISTER

SF	ZF	x	AF	x	PF	x	CF

- SF → Sign flag: 1 → MSB of result 1; 0 → MSB of result 0.
- ZF → Zero flag: 1 = result → 0; 0 = result → 1
- AF → auxillary carry flag: 1: carry from lower nibble to higher nibble; 0: no such carry.
- PF → Parity flag: 1: even Parity; 0: odd Parity
- CF → carry flag: 1: carry out of MSB; 0: no such carry

1) Sign flag

Set : 1 when MSB of result is 1 (result is -ve)
Set : 0 when MSB of result is 0 (result is +ve)

2) Zero flag

It is set when result is 0.
It is reset when result is not zero.

3) AC : Auxillary carry flag

- Set when auxillary carry / Borrow is generated.
- It is reset when an auxillary carry / Borrow is not generated

4) Parity flag

- Set 1 when set has even parity.
- set 2 when result has odd parity.

5) Carry flag

- Set when carry / Borrow is generated from MSB
- It is reset when a carry / Borrow is not generated from MSB.

Interrupt control Pin diagram

1) ~~TRAP~~

f) Timing and control circuit

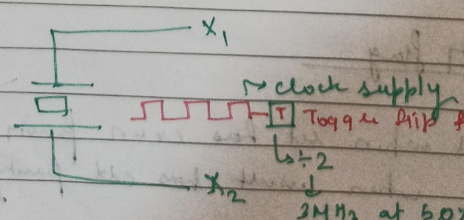
Timing and control circuits issues various internal & external control signals for executing a instruction.

The external pins connected to the circuits are as follows:

a) X_1 and X_2

These pins provide the clock input to the µp.
Clock is provided from a crystal oscillator.

Crystal oscillator of 6MHz : 6MHz



* every clock pulse triggers processor to do a new activity
* speed of processor depends on clock pulse.

frequency clock $\propto$ speed of µP.

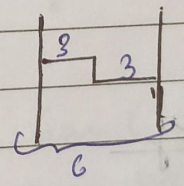
* 8085 µP has a frequency of 3MHz

3MHz : 3 million operations per second.

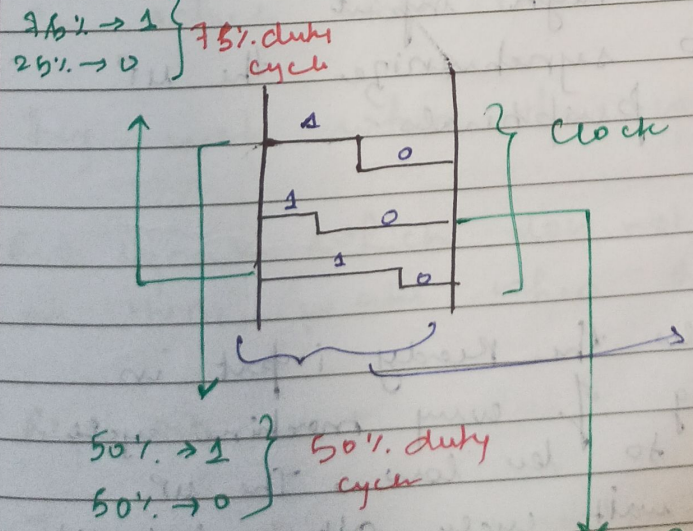
Q) 8085 µP ~~is~~ a has 3MHz frequency but we ~~fe~~ have connected crystal with 6MHz frequency. why?

because 8085 µP needs 3MHz at 50% duty cycle.

* what is duty cycle?



↳ determines the shape of the clock.



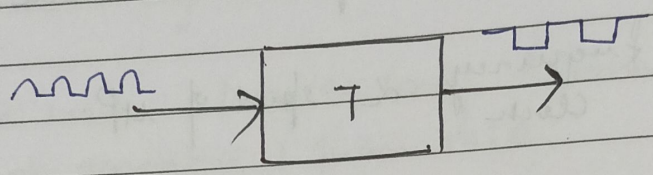
Clock fluctuates at every point; high

Time period same for every duty cycle
↳ frequency also same.

~~NOTE~~:

Toggle flip flop } change of the JK flip flop

ulta karna



b) Clock out

8085 provides the clock input to all other peripherals through the clock out pin.

It takes care of synchronizing all peripherals with 8085.

c) Ready

→ It is an active high input
→ It is used to synchronize the µP with 'slower' peripherals.

Working →

→ The µP samples the Ready input in the beginning of every machine cycle.
→ If it is found to be low, the µP executes one wait cycle after which it resamples the ready pin till it finds the ready pin HIGH.

- The μP remains in wait state until the ready pin becomes high again.

* If the ready pin is not required, it should be connected to V_{cc} and not left unconnected, otherwise would cause the μP to execute infinite wait cycles.

13/06/22.

d) Reset In

- active low signal activated when manual reset signal is applied to the microprocessor.
- This signal resets the μP.
-

* NOTE: on reset, the PC contains 0000H thus, the reset vector address ~~of is 0000H~~ of 8085 is 0000H.

* NOTE: Any signal having a bar (–) over them, means that those signals are active low signals.

They work when the value is 0.

Rest signals which does not have a bar over them, works when they have value 1.

e) Reset out

This signal is connected to reset input of all peripherals.

It is used to reset the peripherals once the µP is reset.

4) ALE

Address latch Enable

This signal is used to latch address from the multiplexed address data bus (AD0 - AD7).

When bus contains address, ALE is high. Else it is low.

$A_0 - A_{15}$

$A_{15} - A_8$ — always address

$A_7 - A_0$ / $D_7 - D_0$ — multiplexed

~~In any operat~~

$AD_7 - AD_0$ } sometimes, address, sometimes data

In any operation, we do not give address and data together. We give address first and then data. So, we can send ~~this~~ data and address together through same bus and ~~our~~ the number of pins will reduce. That is why

we multiplexed the address and data bus.

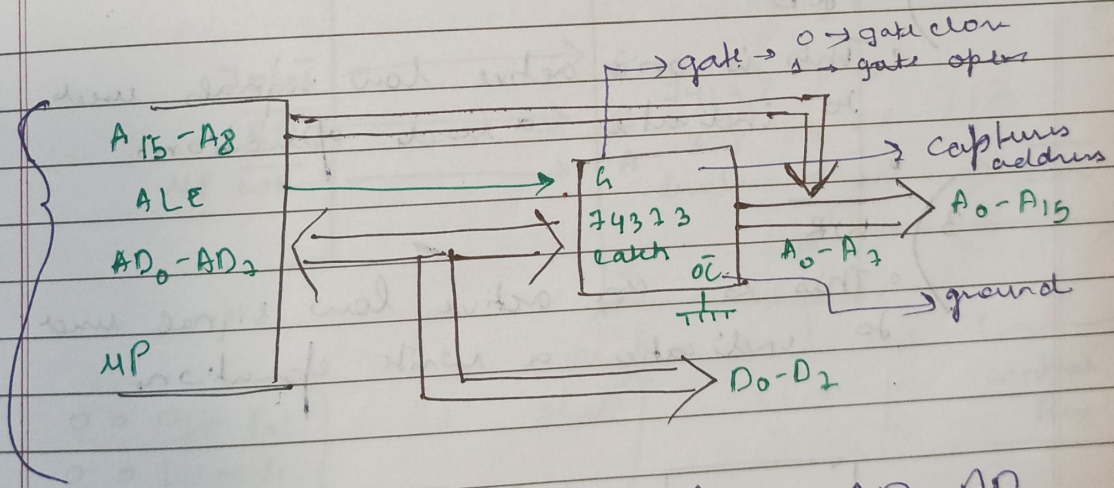
↓

Means combing

* How would we know if its data or address from $AD_7 - AD_0$?

ALE helps in identifying weather its data or address.

If ALE = 1 → address in the bus
ALE = 0 → data in the bus.



→ seperated Adress and data from $AD_0 - AD_7$ multiplexed bus.

OR

latching of address from multiplexed address bus.

* ① $AD_7 - AD_0$: always bidirectional.
~~If~~ This bus has address as well as data.
~~We can~~ address can ~~at~~ only be read. but,

we can read as well as write the data. Thus, bidirectional.

g) <u>Control signals</u>

1) $IO/\overline{M}$

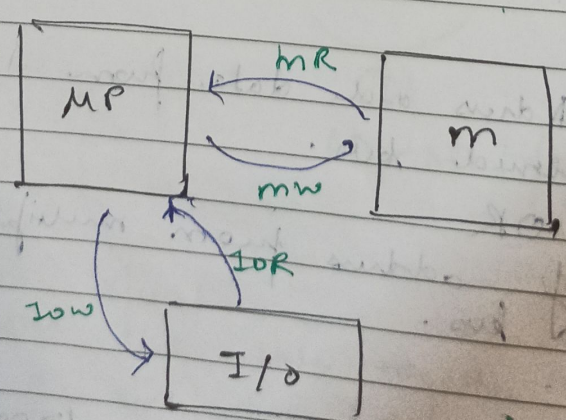
- This signal is used to distinguish between IO and a memory operation.
- when this signal is high, then it is an IO operation else it is a memory operation

2) $\overline{RD}$

- This is an active low signal used to indicate a read operation.

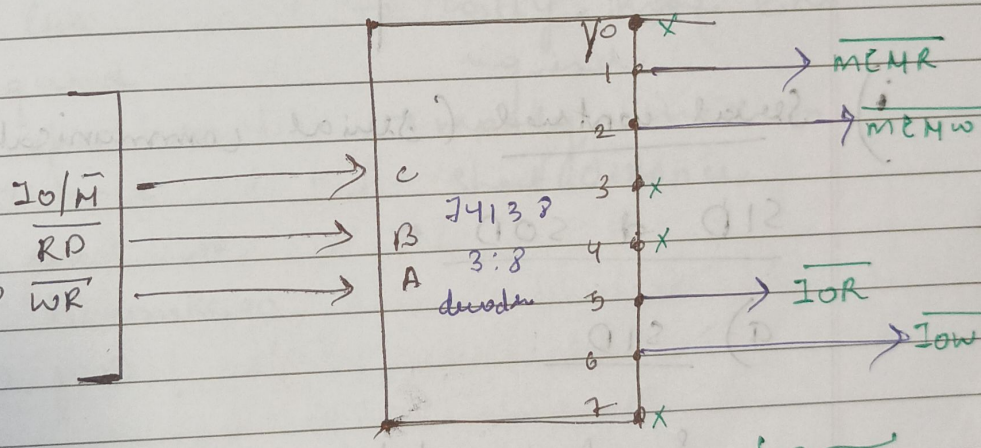
3) $\overline{WR}$

- This is an active low signal used to indicate a write operation.



	$IO/\overline{M}$	$\overline{RD}$	$\overline{WR}$	
MR	0	0	1	→ Y_1
MW	0	1	0	→ Y_2
IOR	1	0	1	→ Y_5
IOW	1	1	0	→ Y_6

$IO/\overline{M}$ when 1 : IO
when 0 : $\overline{M}$



control Bus

0 0 0 – Y_0
0 0 1 – Y_1
0 1 0 – Y_2
0 1 1 – Y_3
1 0 0 – Y_4
1 0 1 – Y_5
1 1 0 – Y_6
1 1 1 – Y_6

h) Status signals

S_1 & S_0

These lines denotes the status of uP.

S_1	S_0	Status
0	0	Idle
0	1	write
1	0	Read.
1	1	opcode fetch.

i) Serial control (serial communication).

SID & SOD

a) SID :

Serial in data
up receives data, bit by bit through this line.

b) SOD : Serial output data

→ uP sends out data bit by bit through this line
→ Serial transmission can be done by RIM and SIM instructions.

Communication

Serial	parallel
→ sends bit one-by one	→ sending multiple bits together. Ex: our data bus is 8 bits. It sends 8 bits data together.
→ slow	→ faster
→ cheap, less lines required	→ costly, more lines required.
→ mostly used in long distance	→ mostly used in short distance
→ also used in wireless transmission	
→ Ex: SID, SOD	

j) V_{cc} & GND

'5' in 8085 signifies 5 volt power supply

k) Interrupt control

This block is responsible for controlling the hardware interrupts of 8085.

1) TRAP (|←)

→ This is an edge as well as level triggered, vectored interrupt.
→ It cannot be masked by SIM instruction and can neither be disabled by DI instruction.
→ It has the highest priority.
→ Its vector address is 0024 H.

2) RST 7.5 (RST means Restart) (|←)

→ Edge triggered vector interrupt.
→ can be masked by SIM instruction and can also be disabled by DI instruction.
→ It has second highest priority.
→ Its vector address is 003C H.

3) RST 6.5 (|←)

→ level triggered vector interrupt.
→ can be masked by SIM instruction and can also be disabled by DI.
→ It has 3rd highest priority.
→ Its vector address is ~~003CH~~ 0034 H

Evergreen
Page No.
Date: / /20

4) RST 6.5.5 (I←)

→ This is a level triggered vector interrupt
→ It can be masked by SIM instruction.
→ and can also be disabled by DI
→ It has 4th highest priority.
→ Its vector address is 002CH.

5) INTR (I←)

→ This is a level triggered, non vectored interrupt.
→ It cannot be masked by SIM instruction but can be disabled by DI instruction.
→ It has lowest priority.
→ It has an acknowledgement signal $\overline{INTA}$.
→ The address for ISR is fetched from external hardware.

6) $\overline{INTA}$ (I→) Interrupt acknowledgement

up gives acknowledgement signal to interrupt signal.

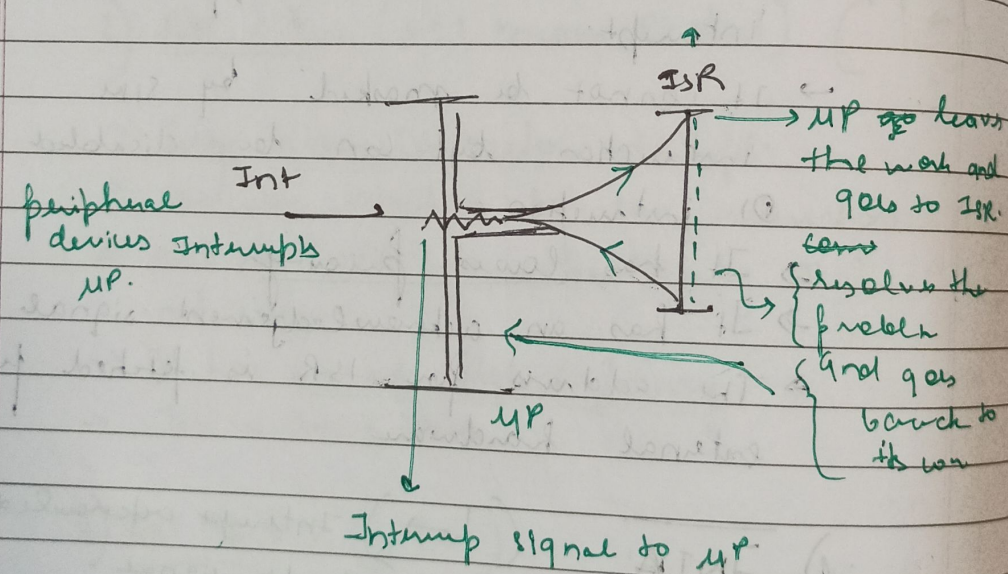
→ This is an acknowledgement signal for ~~INTR~~ INTR only.
→ This signal is used to get the op code (and hence the ISR address) from internal hardware in order to execute the ISR.

* All Interrupts except traps can be disabled through DI instruction.

Then ~~instrut~~ interrupts can be enabled again by EI instruction

Interrupts can be individually ~~as~~ masked or unmasked by SIM instruction.

TRAP & INTR are not affected by SIM instruction.



e) HOLD & HLDA

→ The HOLD & HOLD acknowledgement signals are used for direct memory access. (DMA)

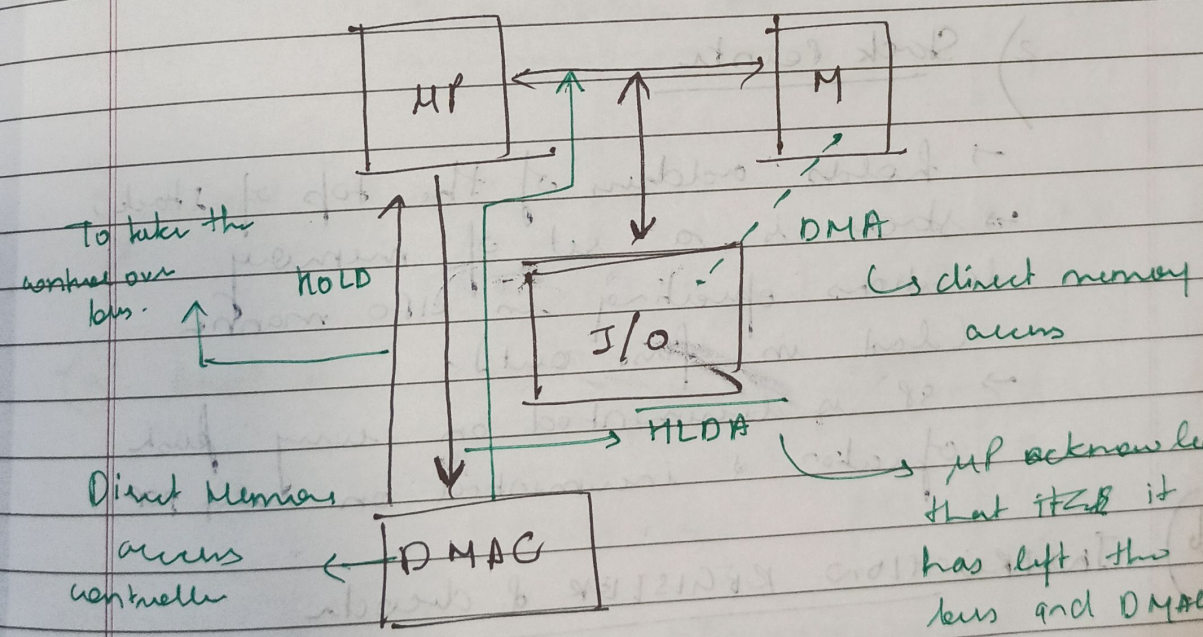
→ The DMA controller issued the HOLD signal to the μP.

→ In response, the μP releases the ~~signal bus~~ system bus.

→ After releasing the system bus the μP acknowledges the HOLD signal with HLDA signal.

→ The DMA transfer thus begins.

→ DMA transfer is terminated by releasing the HOLD signal.



HOLD → 1 : asking for control
HOLD → 0 : not asking for control anymore.

Architecture

(a) Register

1) Program counter

→ 16 bit special purpose register
→ holds address of next instruction
→ PC is incremented by INR/DCR after every instruction byte is fetched.

2) Stack Pointer

→ holds address of the top of stack,
→ stack is a set of memory locations operating in LIFO manner (last in first out).
→ SP is decremented on every push operation & incremented on every pop.

b) INSTRUCTION REGISTER & decoder

1) Instruction Register

→ 8085 places the contents of PC onto the address bus & fetches the instruction.
→ The fetched register is stored into the instruction register.

Evergreen
Page No.
Date : / /20

2) Instruction decoder

→ The fetched instruction from the instruction register enters the instruction decoder.

→ Then, the instruction is decoded and decoded instruction is given to timing & control circuit. When the instruction is executed.

Instruction set

3) Logic group

(a) ANA R

Eg: ANA B

$A \leftarrow A \wedge B$

AND	
0 0	0
0 1	0
1 0	0
1 1	1

(b) ANA M

$A \leftarrow A \wedge M$

Clear NB:

A =	0011	0101 235
ANI, OFH	1111	0000
	0011	0000

Timing diagram 8085

Timing diagram.

Instruction cycle:

This is the time required by µP to fetch and execute one complete instruction.

The instruction cycle is in two parts.

(1) Fetch cycle
(2) Execute cycle.

(a) Fetch cycle: This is the time required by the µP to fetch all bytes of an instruction.
The length of the fetch cycle is thus determined by the no. of bytes in an instruction.

(b) Execution cycle:

This is the time required by the µP to execute a fetched instruction.

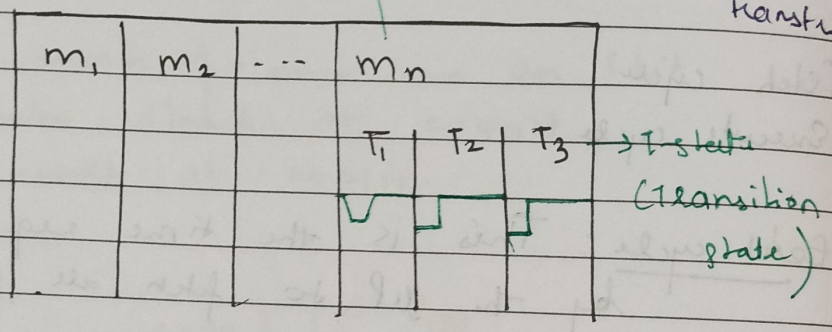
→ An instruction cycle consists of various machine cycles.

→ The most important and also compulsory machine cycle of every instruction is OPCODE FETCH.

→ There are other machine cycles such as memory read, memory write, I/O read etc.

→ Machine cycles contain T-states.

fetch + decode + execute. ← { Instruction cycle

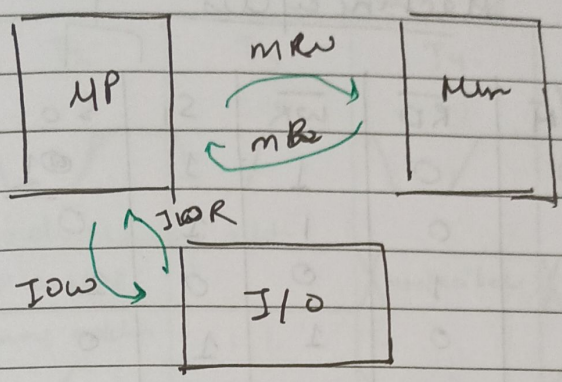


Machine cycle :

→ Time required by Microprocessor to complete an operation of accessing memory or input / output devices is called Machine cycle

→ As the data bus of 8086 is 8-bit, one machine cycle will transfer one-byte. (8 bits)

→ Instruction that require to read or write 16-bit data from memory, need multiple machine cycles.



T-state

a t-state is ~~one~~ one clock cycle of the µP.

∴ T = clock period = 1/clock frequency

- since 8085 works at 3MHz,
  one T-state = 1 ÷ 3 microseconds
  = 0.333 microseconds

∴ a processor calculates time in clock pulses, not in seconds or minutes.

→ every clock pulse triggers processor.

Input / output

Machine cycles

	IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	S_1	S_0
OF*	0	0	1	1	1
MR	0	0	1	1	0
MW	0	1	0	0	1
IOR	1	0	1	1	0
IOW	1	0	1	0	1

→ 1 becomes if so becomes 0, the OF and MR becomes same. (~~OF & MR are~~ different becom ~~OF~~ ~~to~~ OF, decode as well)

opcode fetch.

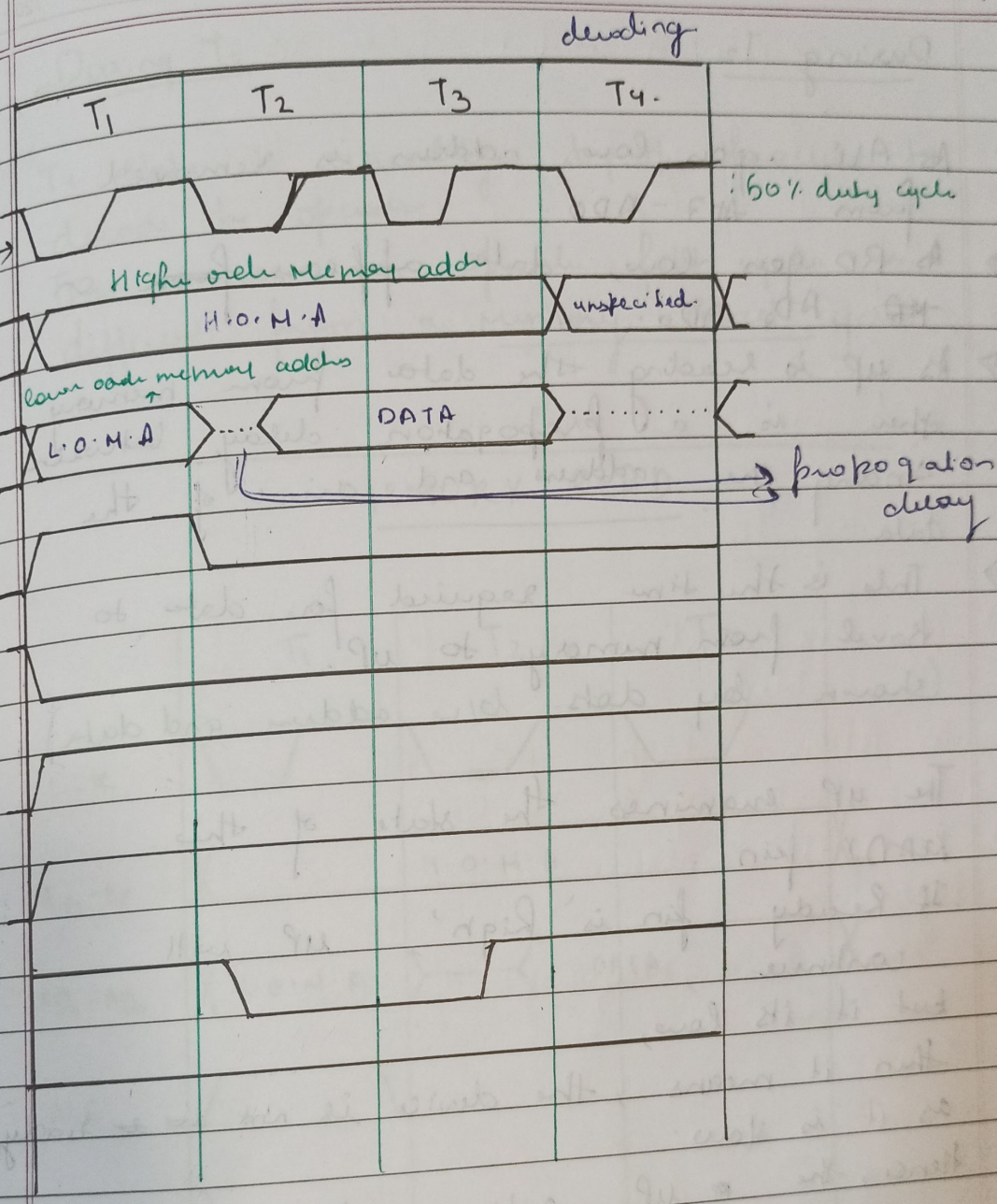
∴ opcode is fetched from the memory

∴ MP reads ~~OP~~ opcode from memory

	T-state
OF	4T/6T → 4state → decode.
MR	3T
MW	3T
IOR	3T
IOW	3T

1st Tstate → MP gives address
2nd → reads
3rd → MP captures data.

OPCODE FETCH



→ During T_1 :

→ $A_{15} - A_8$ contains the higher byte of the address.

→ As ALE is high, $AD_7 - AD_0$ contains lower byte of the address.

→ S_1 & S_0 go high

→ Mem operation: $IO/\bar{M}$ goes low.

During T_2:

- As ALE goes low, address is removed from ~~AD7~~ - AD0.
- As RD goes low, data appears ~~from~~ on ~~KA~~ AD_7 - AD_0.
- As uP is reading the data from memory there is a propogation delay between sending the address and arrival of the data.
- This is the time required for data to travel from memory to uP.
  (shown by dots b/w address and data)

- The uP examines the state of the READY pin.
  If Ready pin is 'high', uP will continue,
  but if its low,
  then it means, the device is not ~~so~~ ~~to~~ ready as it is slow.
  Hence, the ~~uP~~ uP enters wait-state by executing wait cycles and remains in the wait state until Ready goes high.

During T_3:

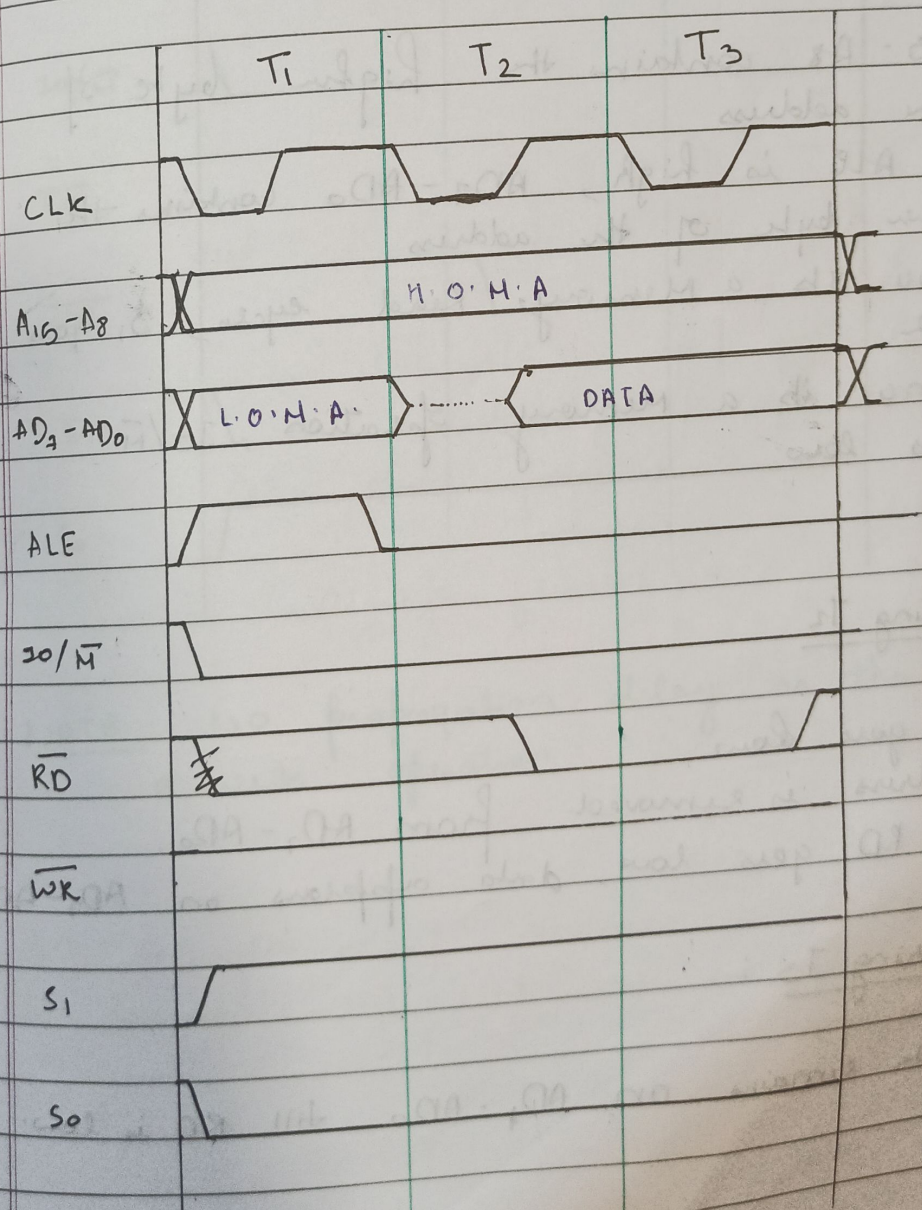
Data remains on ~~D+~~ AD_7 - AD_0 till RD is low. This is the time given to uP to capture the data from the data bus.

During T_4 :

→ T_4 state is usually by the processor to decode the opcode

→ This is how an opcode fetch cycle is different from a memory Read cycle

MEMORY READ



→ This cycle is used to fetch one byte from memory
→ This cycle can be used to fetch the operand bytes of an instruction or any data from the memory.
→ It is a not compulsory machine cycle
→ It requires 3 T states.

<u>During T1</u>

→ A15 - A8 contains the higher byte of the address
→ As ALE is high, AD7 - AD0 contains the lower byte of the address
→ Since its a Memory Read cycle, S_1 goes high
→ Since its a Memory operation, $IO/\overline{M}$ goes low.

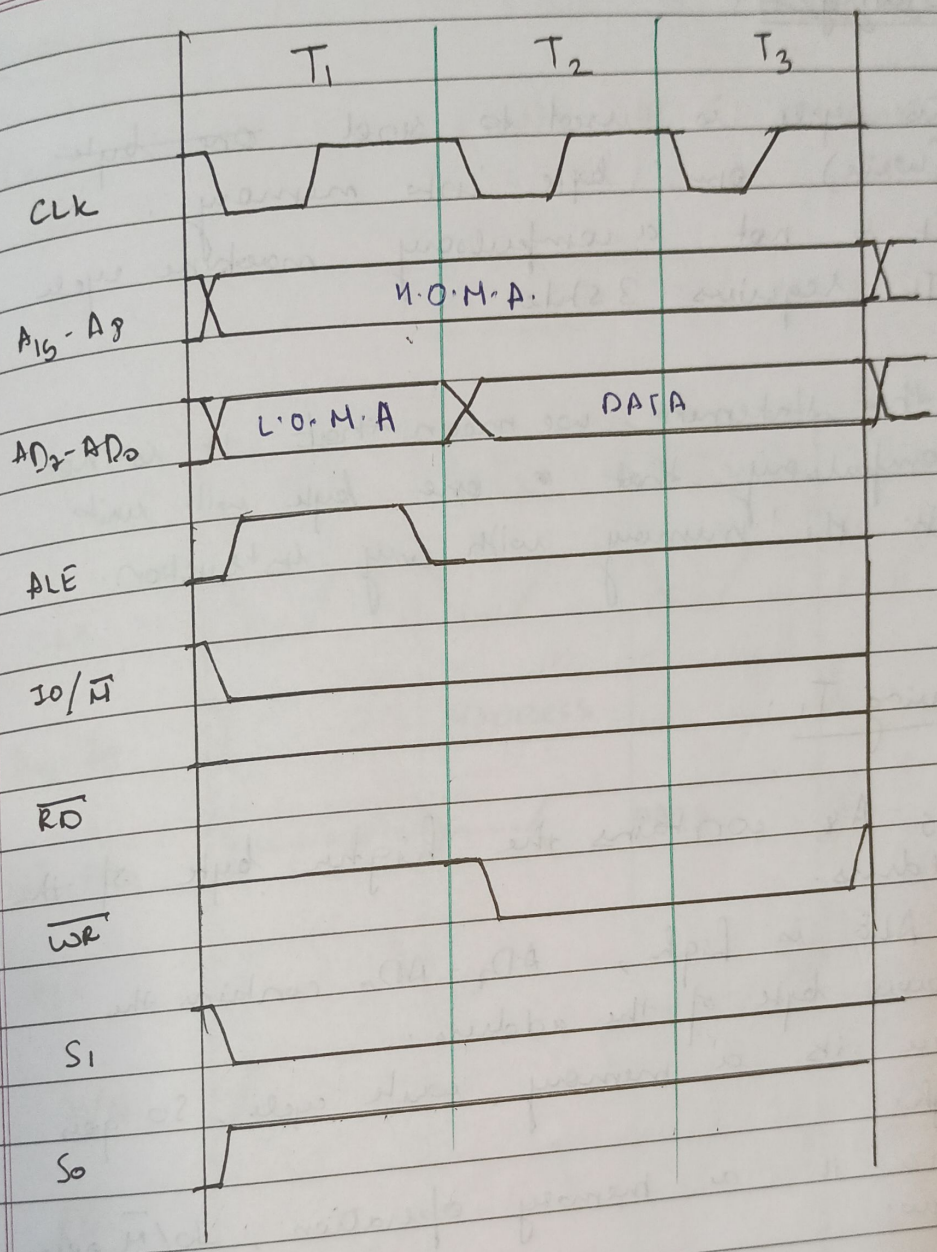
<u>During T2</u>

→ ALE goes low
→ Address is removed from AD7 - AD0.
→ As RD goes low, data appears on AD7 - AD0.

<u>During T3 :</u>

→ Data remains on AD7 - AD0 till RD is low.

MEMORY WRITE



* NOTE: No propogation delay as this is a write operation.

~~During T1~~:

→ This cycle is used to send ~~one byte~~ (write) one byte into memory.

→ It is not a compulsory machine cycle

→ It requires 3 state

→ By this statement, we mean that it is not compulsory that one byte will write in the memory with every instruction.

During T_1:

→ A_{15} - A_8 contains the higher byte of the address.

→ As ALE is high, AD_7 - AD_0 contains the lower byte of the address.

→ Since its a memory write cycle, so goes high.

→ Since its a memory operation, $IO/\overline{M}$ goes low.

During T_2

→ ALE goes low.

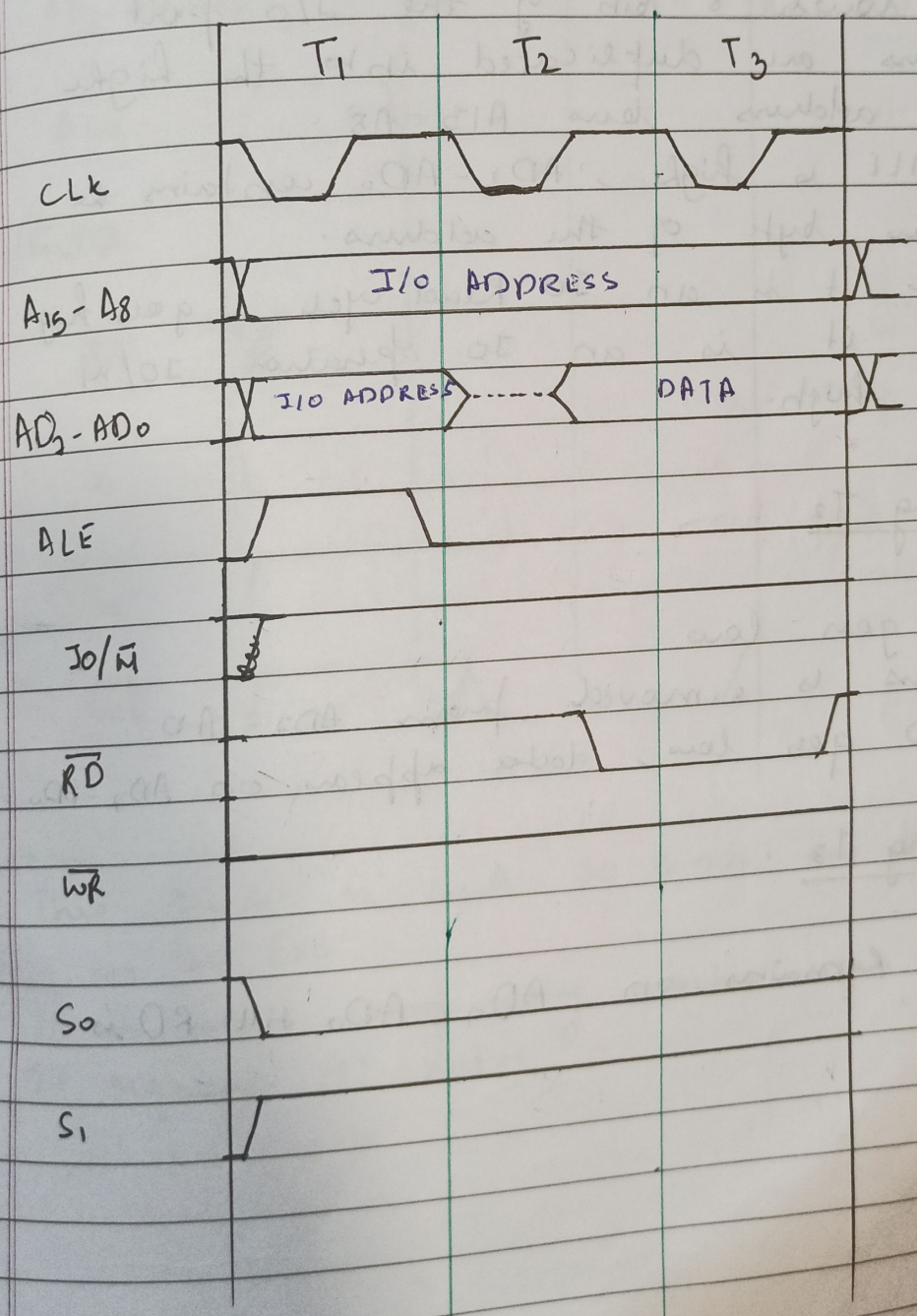
→ Address is removed from AD_7 - AD_0

→ data appears on AD_7 - AD_0 and WR goes low.

During T_3 :

Data remains on AD_7 - AD_0 till $\overline{WR}$ is low.

IO READ



→ This cycle is used to fetch one byte from an IO port.
→ It is not a compulsory Machine cycle
→ It requires 3T states.

During T1

→ The lower 8 bits of the I/O port address are duplicated into the higher order address lines A15 - A8.
→ As ALE is high, AD7 - AD0 contains the lower byte of the address.
→ Since it is an IO Read cycle, S1 goes high
→ Since it is an IO operation, IO/M goes high.

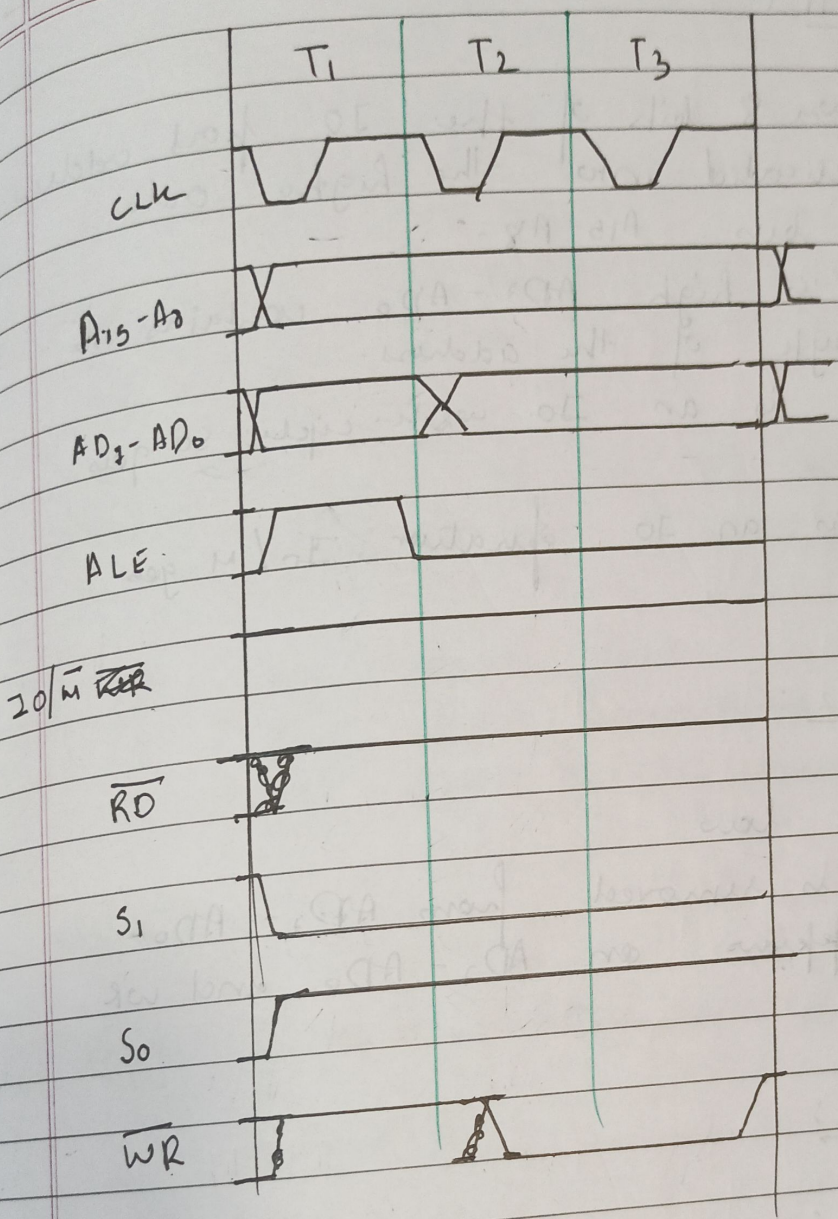
During T2 :

→ ALE goes low
→ Address is removed from AD7 - AD0.
→ As RD goes low, data appears on AD7 - AD0.

During T3 :

→ Data Remains on AD7 - AD0 till RD is low.

IO WRITE



- This cycle is used to send one byte into an IO Port.
- It is not compulsory machine cycle
- It requires 3T-states.

During T_1:

→ The lower 8 bits of the IO port address are duplicated into the higher order address bus $A_{15} - A_8$.

→ As ALE is high $AD_7 - AD_0$ contains the lower byte of the address.

→ Since, it is an IO write cycle; so goes high.

→ Since it is an IO operation, IO/M goes high.

During T_2:

→ ALE goes low

→ Address is removed from $AD_7 - AD_0$.

→ data appears on $AD_7 - AD_0$ and WR goes low.

During T_3:

→ Data remains on $AD_7 - AD_0$ till RD is low

Timing Diagram → Instruction set

→ In ~~ara~~ an Instruction Timing diagram, there are two types of cycles.

1. Fetch cycle
2. Execution cycle.

How to determine the no. of fetch cycle to be executed in an instruction?

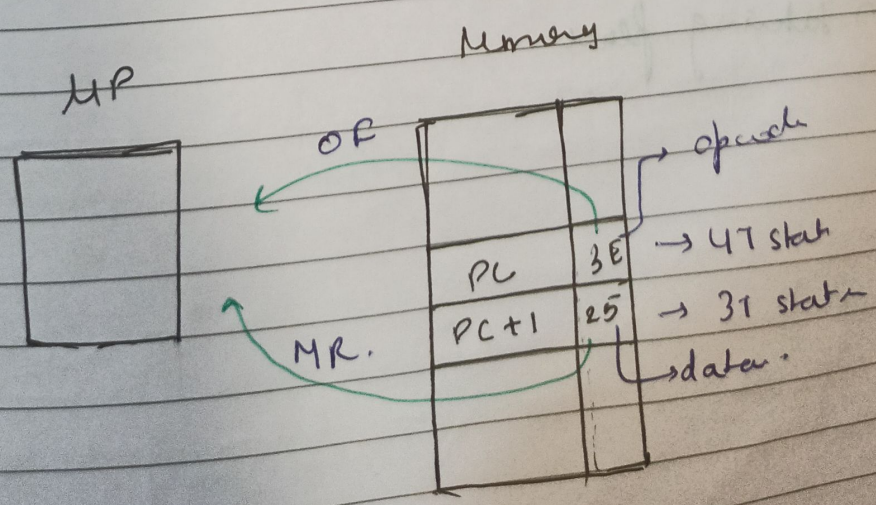
→ 1 Byte : opcode fetch
→ 2 Byte : opcode fetch + Memory read
→ 3 Byte : opcode fetch + 2 Memory read.

~~How to~~.

(1) MVI B, ~~25~~ H : 2 Byte.

B ← 25H

opcode fetch + Memory Read.



(2) LXI B, 2000H 3 Byte

BC ← 2000H

				T state
opcode fetch	PC	opcode fetch		4
Memory Read	PC+1	00		3
Memory Read.	PC+2	20	+	3
				10

(3) LDA 2000H.

A ← [2000H]

opcode fetch	PC	opcode	4T
Memory Read	PC+1	00	3T
Memory Read	PC+2	20	3T
Memory Read	2000	[2000]	3T
			13T

reading the contents of memory location [2000H].
execution taking place.

(4) STA 3000H

[3000H] ← A

opcode fetch	PC	opcode	4T
Memory Read	PC+1	00	3T
Memory Read	PC+2	30	3T
Memory ~~Read~~ write	[3000]	A	3T
			13T

(5) LDAX B

A ← [B-C]

opcode fetch	PC	opcode	4T
Memory Read	BC	[BC]	3T
			7T