Ch-1: Introduction

1.1. Discuss Microprocessor.

Microprocessor is a digital ckt constructed by using LSI (Large Scale Integration), VLSI (Very large scale Integration) and above packages consists of thousand of components in a small 'Si' chip.

Definition:

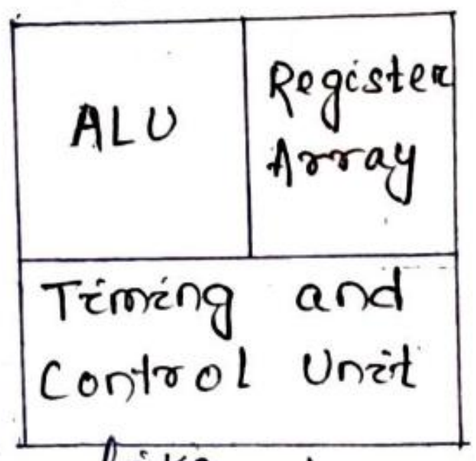
Microprocessor is a multipurpose, programmable, clock driven, register based electronics digital ckt that is

1. Reads the instruction from the storage device called memory.
2. Accepts the binary data as i/p.
3. Process the binary task according to the instruction.
4. Execute the result as o/p.

Organisation of Microprocessor:

Microprocessor consists of different stages, such as

1. ALU (Arithmatic & Logic Unit)
2. Control Unit
3. Register array.



ALU

It performs arithmatic operation like +, −, etc and Logic operation like OR, AND, NOT etc. This is the main part of the µp which compute all the operations.

Control Unit (CU)

This is the timing and control unit of the µp which provides the timing options for the transfer of instruction and datas. This maintains time gap betn the instructions.

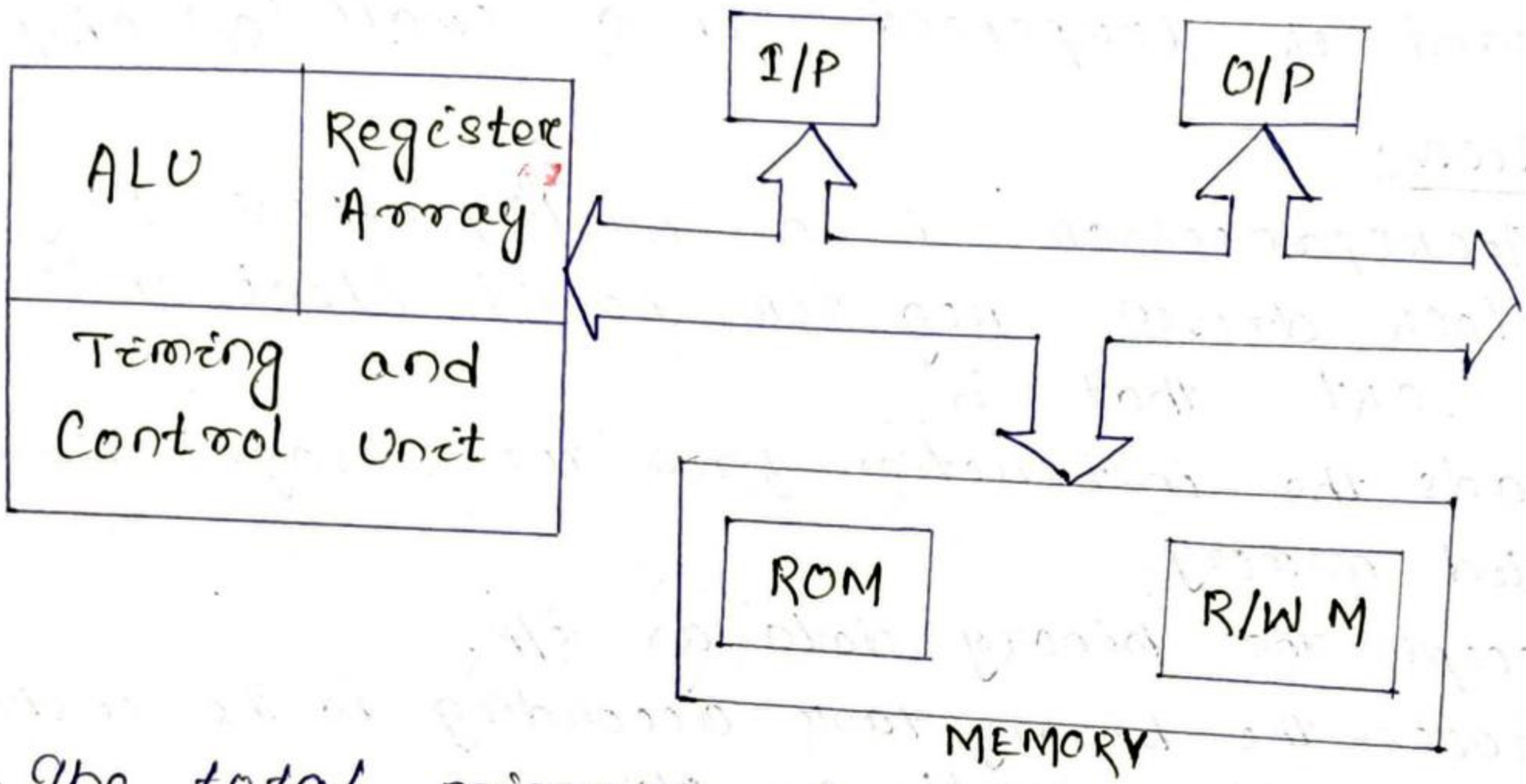
→ This is the unit which enabled on disabled to perform the operation according to the instruction given

Register Array

These are the general purpose registers present inside the µp to store datas temporarily.

→ It helps the µp to compute operations by receiving or transfering the instructions present in this.

Organisation of microcomputer (Digital Computer)



1. The total microcomputer system is excluding all the peripherals is called the system.
2. The individual component connected to perform the task is called subsystem.
3. The line which is provided to the communicative path from µp to other peripherals is called system bus.

I/P

This is the unit which transfers binary instructions from the i/p device to the µp.
Eg: Keyboard, Mouse, CD-ROM, Joystick.

O/P

This is the unit which transfers binary informations from µp to O/P device.
Eg: Monitor, printer, Speakers, LCD scree/Projector.

Memory

This is the unit which stores informations and instrs in the form of binary digit. It is of two types i,e ROM & R/W M.

ROM (Read Only Memory)

ROM is used to store instructions permanently. By using this memory, computer can execute the initial display.

To communicate with the peripherals, the MPU performs the following operations.

1. Identify the peripheral or memory location (Address)
2. Transfer of binary information (Data or instruction).
3. Sending of control or synchronised signal.

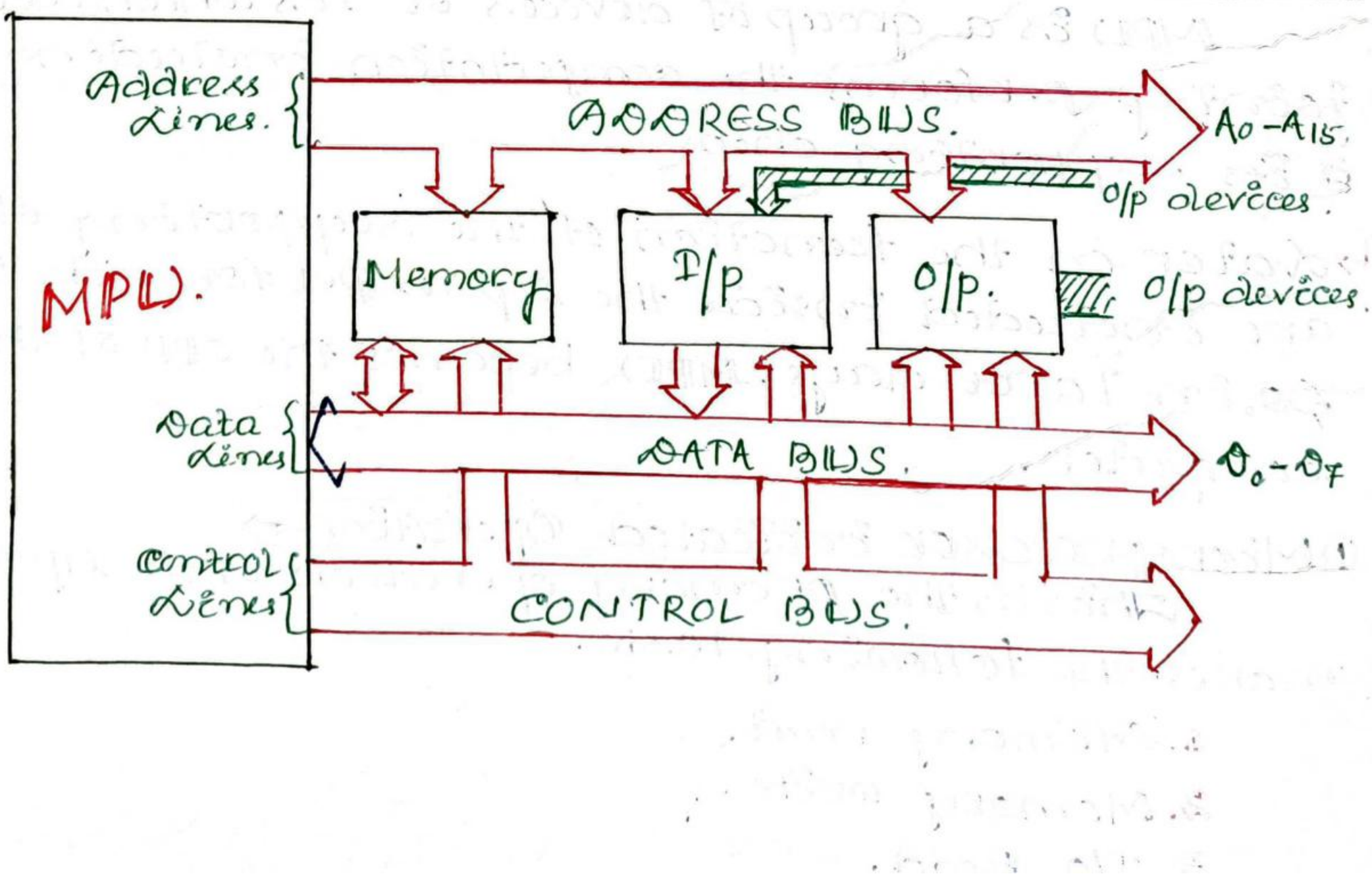
2.2. State & Explain Buses.

To perform these operations, MPU needs some communicating path called system bus. The system bus is divided into 3 parts according to its communication.

1. Address bus.
2. Data bus.
3. Control bus.

2.3. Study general bus structure.

Bus structure of 8-bit µp →



2.4. Describe address bus, data bus, control bus.

Address bus →

It is the group of lines used by the µp to identify the peripherance or the memory location present inside the memory device.

↳ Because the address was send by the µp to others for identification, it is of unidirectional.

↳ Each device & location are assigned by a binary no. which is the address of that location.

↳ 8085 µp has 16 address lines denoted by A_0-A_{15}.

* 8086 µp has 20 address lines denoted by A_0-A_{19}.
* ~~8085~~ 80286 µp has 32 address lines denoted by A_0-A_{31}.

Data bus →

This is the group of lines used to transfer data or instructions from µp to other peripheral & vice versa.

↳ Because datas are flow from µp to others & vice versa. So, it is a bidirectional lines.

↳ 8085 µp has 8 data lines denoted by D_0-D_7.
8086 µp has 16 data bus denoted by D_0-D_{15}.

80386 } (Pentium) has 32 data lines denoted by D_0-D_{31}.
80486 }

Control Bus →

It is the single line used by the µp to control different operations.

↳ Every operation is assigned a specific control signal.

↳ It is ~~also unidirectional~~ partially unidirectional & partially bidirectional.

↳ µp sends a pulse to activate the particular operations

e.g. (i) Memory read - $\overline{MEMR}$.
(ii) Memory write - $\overline{MEMW}$.
(iii) I/O Read - $\overline{I/OR}$
(iv) I/O write - $\overline{I/OW}$.

↳ For I/O read operation, 1st µp sends the addressing signal to the I/O device.

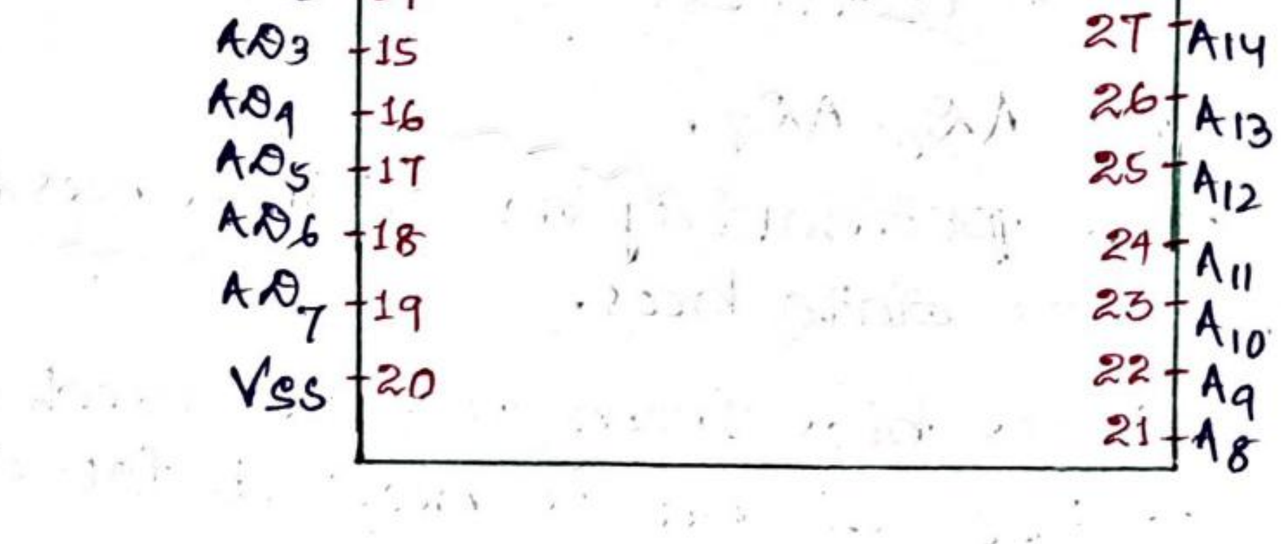
↳ The address send by the µp is decoded by an external logic ckt which makes the activation of I/O location.

↳ Then µp sends a controlled signal $\overline{IOR}$ / $\overline{IOW}$ by sending a pulse to the I/O chip which make activate that perticular I/O location from which the user wants to read.

↳ Once the I/O chip is activated then the datas are flow from the I/O location to µp through bidirectional data bus for computation & execution.

2.5. Describe pin structure of 8085 µp.

PIN DIAGRAM OF 8085 µp →



8085A Pin dia

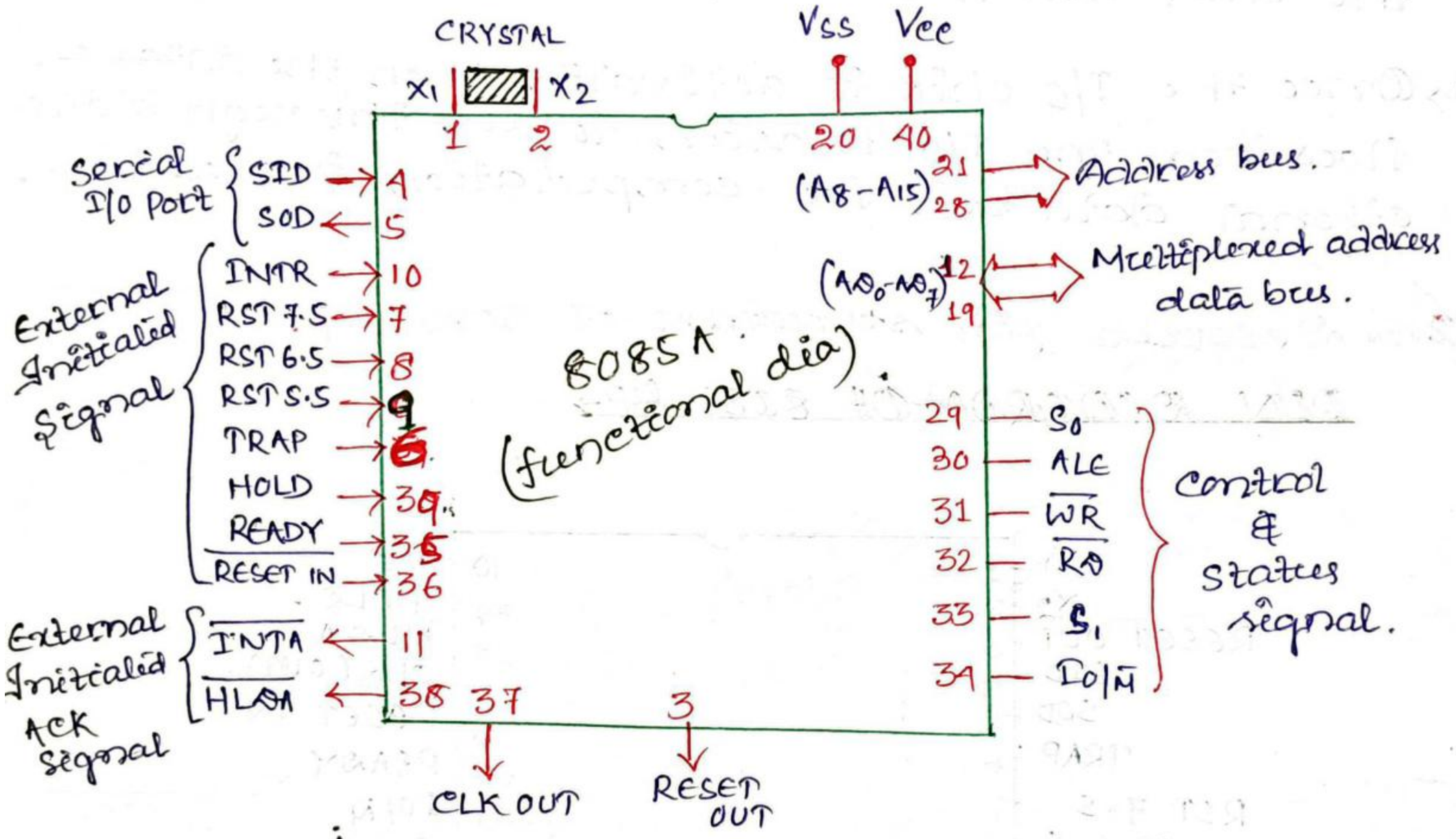
The total pin operation of the 8085 µp which is similar to 8085A are divided into following parts:

1. Address bus.
2. Multiplexed address / data bus.

3. Control & states signal.
4. Power supply & frequency signal.
5. External initialed signal.
6. Serial I/O port.

1. Address Bus →

8085 µp has 8 high order address bus denoted by A_8 to A_{15}.



2. Multiplexed Address/Data bus →

8085 µp has dual purpose address & data bus called multiplexed address data bus.

- It is denoted by AD_0-AD_7.
- µp uses this pins primarily as address bus & then it can be used as data bus.
- To separate address bus from multiplexed address, data bus, a control signal is needed for demulti-plexer.
- These are low order address bus.

3. Control & Status signal →

Mp uses this signals to locate or identify the memory location & this signal show the operation of the Mp.

ALE → (Address latch Enable).

This is the control signal used by the Mp to demultiplex the low order address bus from multiplexed address data bus.

↳ This signal enables the address bus to be used by the Mp A_0 to A_7.

$\overline{WR}$ → (Write).

This is the control signal used by the Mp to write the datas or informations in the memory loc. ↳ This control signal enables the memory chip where the user wants to write.

$\overline{RD}$ → (Read).

This is the control signal used by the Mp to read the datas or informations from the memory location.

↳ This control signal enables the memory chip where the user wants to read.

IO/$\overline{M}$ → (Input output/Memory) →

- This is a type of control signal used by the Mp to identify wheather the operation is related to I/P, O/P device or memory device.

↳ When this pin is in high state it identifies the I/O operation.

↳ When it is low, it identifies the memory operation.

↳ It identifies the memory location.

↳ This pin will operate with the collaboration of $\overline{RD}$ or $\overline{WR}$ control signal.

$S_0, S_1 \rightarrow$

This is the status signal used by the µp to identify the status of different operation.

S_0	S_1	Result
0	0	Halt
1	0	$\overline{WR}$
0	1	$\overline{RD}$
1	1	$\overline{INTA}$

1. Power supply & frequency signal →

This pins are provided on the µp to supply dc voltage, grounds, frequency for the µp.

Vcc →

8085 µp requires +5V dc power supply.

Vss →

8085 µp has a pin which is connected to the ground.

X_1 & X_2 →

This is the two point between which a crystal is connected to provide 3MHz clock frequency to internal pins.

↳ The frequency is internally divided into two equal halves. So, the frequency supplied by the crystal will be 6MHz.

CLK (out →

It is the clock signal generated by the µp for the other devices.

5. External Initial signals including Interrupts →

This is the group of signal used by the external peripherals with the 8085 µp.

1. INTR → (Interrupt request) →

↳ It is the signal sends by the external device to interrupt the current execution.

→ It is a request signal from the external devices to the µp.

2. $\overline{INTA}$ → (Interrupt Acknowledgement) →

↳ This is an outgoing signal generated by the µp with the request of external device INTR.

↳This signal is a granted signal generated against INTR.

3. RST 7.5, RST 6.5, RST 5.5 → (Restart signal) →

There are the i/p signal used by the external peripherals for the 8085 µp. for changing the sequence of execution of current operation.

↳Depending upon the priority given by the external devices, the pin goes high or low.

↳The priority order of these pins are 7.5, 6.5 & 5.5.

↳This interrupt signals are the highest priority than INTR.

4. TRAP → (NMI) (Non-Maskable Interrupt) →

This is the highest priority interrupt signal causes during the execution of the program to make the operation suspended.

5. HOLD →

It is the i/p request signal generated by the external peripherals for the µp for the permission of using data & address bus.

↳This system is used by DMA controller to hold the current execution for the time of data transmission.

6. HLDA → (Hold Acknowledgement o/p signal) →

↳This is an outgoing acknowledgement generated against the hold request signal.

↳When this signal is generated the external peripheral have the permission for the data transfer by using data & address bus.

7. READY →

↳This is the signal used by the user which identifies wheather the µp is ready to work or not.

↳When this signal goes low determines that µp is in OFF state.

↳This signal is generated at the time of initial

5. RESET IN →

↳ This is the type of signal sended by the user to make response to the current execution.

↳ When this signal is activated, all the instructions present inside the µp are transfered to the memory location.

9. RESET OUT →

↳ This is the outgoing signal generated by the µp which identifies that the µp being resets.

↳ The signal is also used to RESET the other external devices.

6. Serial o/p port →

There is a provision in 8085 µp to communicate with external peripherals by transferring or receiving datas in a single line.

2.6. Describe Internal architecture of 8085 µp.

Internal Architecture of 8085 µp →

The internal architecture of 8085 µp showing in the fig. is consists of,

1. Arithmatic Logic ckt.
2. Timing & control unit.
3. Instruction register & decoder.
1. Register Array.
5. Interrupt controller.
6. Serial I/O controller.

2.9. Discuss Arithmatic Logic unit.

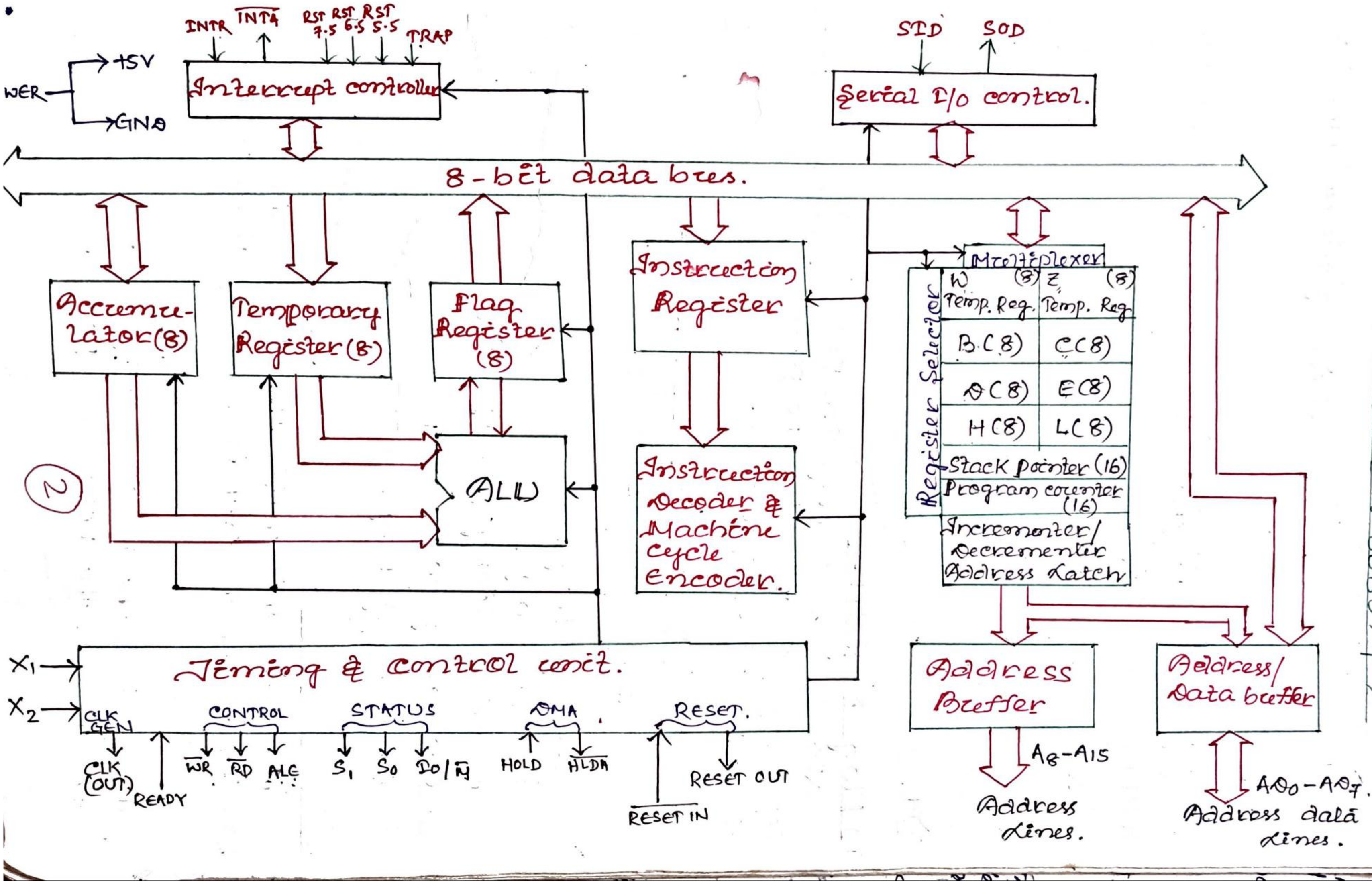
① Arithmatic Logic ckt → (ALU) →

The arithmatic logic unit is present inside the µp to compute all the operations.

The units consist of,

(i) Temporary register (8).
(ii) Arithmatic & Logic ckt.
(iii) Accumulator (8).
(iv) Flag register.

(1)



Architecture of 8085 µp →

Scanned by TapScanner

(i) Temporary Register →
This is a 8-bit temporary storage register used to store the information during the arithmatical & logical operation.

(ii) Arithmatic & Logic unit →
This is the ckt provided inside the µp to perform arithmatical & logical operation.

(iii) Accumulator →
It is a 8-bit register used for the storing of the result obtained during the arithmatical & logical operation inside the ALU.

2.14. Describe flag register.

(iv) Flag Register →
This is a 8-bit register provided inside the µp only set of 5 f/fs are used to test the condition of the result of operation.

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

S → Sign flag.
Z → Zero flag.
AC → Auxiliary carry flag.
C → Carry flag.
P → Parity flag.

Sign flag (S) →
After the arithmatic operation, if the left most bit (D_7) is SET or RESET is identify about the sign bit.
(i) If D_7 is 1, the result is negative.
(ii) If D_7 is 0, the result is positive.

(3)

Zero flag (Z) →

Ocering the ALU operation, if the result of ocend is zero, then the 7th bit (O_6) is SET.

(i) If O_6 is 1, the result is 0.

(ii) If O_6 is 0, the result is 1.

Auxelliary carry flag (Ac) →

Ocering the ALU operation, if a carry bit is generated in 4th position (O_3) & passed on to the 5th position (bit) O_4 then it saeid to be generation of auxelliary carry.

↳ This is internally present to perform BCO operation.

↳ This is not available for the programmer to change the stat.

Parity flag (P) →

Ocering the ALU operation, if the result contains even no of '1' then the parity flag is SET.

(i) If O_2 is in 1 then the result is having even no. of 1s.

(ii) If O_2 is in 0 then the result is having odd no. of 1s.

Carry flag (C) →

Ocering the ALU operation, for addition of multiplication, if a carry bit is generated then a carry flag is SET.

(i) If O_0 is 1, the result is having a carry bit.

(ii) If O_0 is 0, the result is not having a carry bit.

↳ For a subtraction operation, this flag can be used as a borrow flag.

(4)

2. Timing & control unit →

Timing & control unit synchronise, the up with a clock signal to commenicati betn. up & other Peripherals.

↳ The control signal used by the up are, $\overline{WR}$, $\overline{RD}$,

ALE to identify about the operation.

↳ This reset is also having DMA option to communi-cate b/w the I/O device & memory without the help of µp (HOLD, HLDA).

↳ This reset is also having RESET signals RESET IN & RESET OUT to initialise the program & to RESET the external devices.

↳ This reset is also having a clock generator (CLK OUT) to provide CLK signal of 3MHz freq. to the external devices & the CLK signal is provided by a crystal of 6MHz freq. b/w pins X_1 & X_2.

(3) Instruction Register & decoder →

When an instruction is executed from memory, it is stored in instruction register.

↳ It stores the instruction temporarily & send to the instruction decoder.

↳ The instruction decoder decodes the instructions & identifies what program or operation should be followed.

↳ It is an internal register can't available for the programmer to change the contents.

↳ It is a part of ALU.

(4) Register array →

↳ 8085 µp has 6 general purpose register named B, C, D, E, H & L of 8-bit capacity to hold or store data temporarily during the operation.

↳ It has also two temporary register named W & Z of 8-bit capacity to hold the data during execution.

↳ These registers W & Z can't available for the programmer to change the contents.

(5) Interrupt Controller →

Interrupt controller is an unit present inside the

architecture of the µp to control all the interrupt signal coming from the outside of different priority (INTR, RST 7.5, RST 6.5, RST 5.5 & TRAP) by sending an acknowledgement signal $\overline{INTA}$.

6. Serial I/O controller →

It is an unit present inside the architecture of the µp to provide the option of receive & sending the datas in serial form.

2.7. Describe three state registers, three state switches.
2.8. Study the data transfer using tristate registers.

TRISTATE →

When a system utilizes three different conditions using a single line is called tristate.

The states are:- 1. Applied high voltage (Logic-1).
2. Low voltage (Logic-0).
3. Open ckt or undefined state.

Tristate Device →

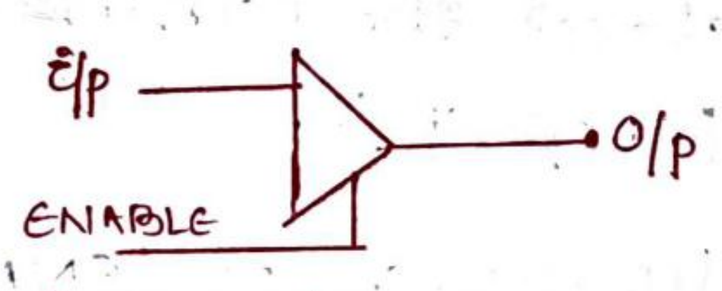
The device which uses three different states of operation. (i) ON state.
(ii) OFF state.
(iii) Electrically disconnected is called tristate device.

Tristate Logic →

A logic ckt having a third o/p line is called "ENABLE" is called tristate logic.

↳ When the ENABLE pin is an active high state then a logic ckt can perform as the ordinary logic ckt.

↳ When the ENABLE pin is in active low state then a logic ckt goes to high impedance state it means no o/p current can drawn across the o/p of logic ckt.



ENABLE PIN	RESULT
Active high	o/p form
Active low	NO o/p.

2.10 Explain program counter.

PROGRAM COUNTER →

It is a 16-bit register used to locate the address of the memory location from which the next operation may be performed.

↳ Because the address of the location is a 16-bit binary no. thats why the length of the register is 16-bit.

2.11. State & explain stack pointer, stack & stack top.

STACK [illegible]TOP →

↳ It is the last location available from the memory location assigned by the stack inside the R/W memory.

↳ It is the top most location from the group of memory locations.

STACK POINTER →

↳ It is a 16-bit register present inside the architecture of µp to point out the specified memory location (stack) inside the R/W memory.

↳ It is loaded by the address of the stack which it is pointing.

↳ This is used to execute operation initially at the time of primary operation.

↳ It is used to point out the stack location.

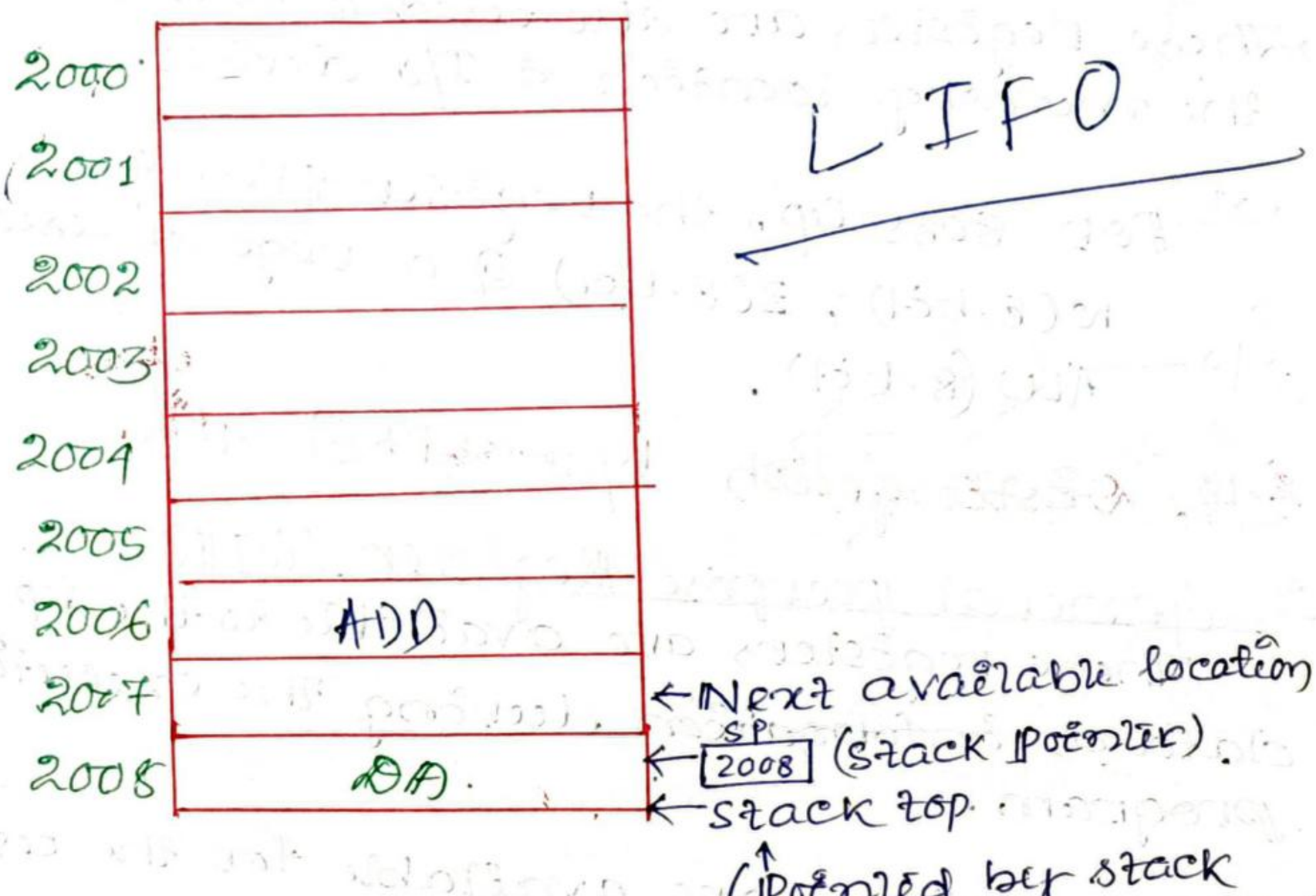
STACK →

Stack is the group of memory location present inside the Rd/$\overline{WR}$ memory which is ~~present~~ previously decided by the programme by writing a program initially.

↳ This is the storing locations used to hold the datas or informations during the execution of program temporarily.

↳ Stack access faster than memory access.

↳ The datas stored in the stack are last in fast out (LIFO) principle.



2.12. State & explain registers.

REGISTER →

Register is a set of flip-flops used to store datas & address of memories present inside the architecture of µp.

↳ Its function is,

1. To store datas or address of memory.
2. To affect the length of the program.
3. To execute the time of the program.
4. To simplify the program.

↳ Depending upon the operations or function at the time of execution, it is divided into 3 parts.

i.e. 1. Temporary Register (TR)
2. General purpose Register (GPR).

Ch-3 # Instruction Set...

3.1. Discuss Instructions.

Instruction is a command given to the computer for specific operation by the given data.

→ Instruction set of the µp is the collection of instructions designed by the manufacture for a specific µp for execution.

→ It is the set of binary words contained '1' or '0' designed by the manufacturer for specific use only.

→ The instructions used in contains two groups
i,e 1. OPCODE
2. OPERAND

OPCODE →

It is the first part of the instruction, where specifies about the task to be perform.

→ Because, it is used to identify the operation by means of a code, it is called operation code or OPCODE.

OPERAND →

It is the second part of the instruction which specifies the data to be operated.

→ Because, it is used to identify the datas which are takes part in the operation, is called operation data or OPERAND.

Techniques used for analysis of data in an instruction

→ 8-bit or 16-bit of data can be directly given to the instruction itself.

→ Address of the memory loc^n can be given to the instruction inside which the data is to be operated is present.

→ In an instr^n, if a single register is specified then the register is taken as the first OPERAND & the other OPERAND can be taken as the accumulator.

→ In an instr^n, if two registers are specified, then the contents of the register can be taken as the data to be operated.

→ In an instruction, if no data is given, then the instr^n has to operate with the accumulator.

→ The instr^n is devided into three types depending upon the mode of operation.
1. Single byte instruction.
2. Two byte instruction.
3. Three byte instruction.

1. Single byte instruction → It is the type of instruction which contains a single byte called OPCODE.
→ Because of the single byte, it is also called single word operation.
→ It requires a single memory location to store.
→ This instruction is used to communicate, one register to other inside the CPU.
→ The communication may takes place between the CPU & memory location is already loaded on the register of the CPU.
Eg: MOV B, C (Move the contents of the register 'C' to Register 'B')

2. Two byte instruction → It is the type of instruction which Contains two byte. 1st byte contains the OPCODE & the 2nd byte contains the OPERAND.
→ It requires two successive memory Location to store.
→ The first instn which is OPCODE & 1st stored in the 1st memory location & the 2nd instn, which is OPERAND is stored in consecute memory Location.
Eg: MVI A, 08 (Move immediately the datas to the accumulator).
OPCODE OPERAND

3. Three byte instruction → It is the type of instn which contains three byte of length. 1st byte is called OPCODE, 2nd byte is called low order OPERAND & 3rd byte is called high order OPERAND.
→ The 2nd & 3rd bytes are specified the address of the memory location where the datas are to be operated.
→ It requires three successive memory Location.
Eg: LDA 2005H (Load the contents of the memory loc 2005H to the accumulator)
OPCODE OPERAND OPERAND
* The instn may be concerned with these 5 factors →
1. 8-bit or 16-bit data
2. 8-bit or 16-bit address
3. Internal register.
4. General purpose register.
5. Memory location.

Groups of 8085 µp

Hence, in other way instruction set is devided into following groups.

(i) Data transfer Group
(ii) Arithmatic Group
(iii) Logical Group
(iv) Branch Group
(v) stack, I/O and Machine control Group.

(i) Data Transfer Group

MOV R_1, R_2 (Move data; Move the contents of one register to another)

$[R_1] \leftarrow [R_2]$; States - 4, flags - None, Addressing - Register. Machine cycle - 1

→ The content of register R_2 is moved to reg. R_1.

Eg:- MOV A, B (Move the content of reg. B to reg. A)

→ Time taken for execution of this instruction is 4 clock period. One clock period is called one state.

→ Here, NO flags is affected.

MOV R, M (Move the content of memory to Reg.)

$[R] \leftarrow [[H-L]]$; states - 7, flags - None
Addressing - Register Indirect, MC - 2.;

→ Here the content of the memory Location, whose address is in H-L pair is moved to reg. R.

Eg:-
```
LXI H, 2000H   (Load H-L pair by 2000 H)
MOV B, M       (Move the content of memory Loc' 2000H to reg. B)
HLT            (Halt)
```

MOV M, R (Move the contents of reg. to memory)

$[[H-L]] \leftarrow [R]$; states - 7, flags - none
Addressing: Reg. Indirect, MC - 2.

→ Here the content of register R is moved to the memory Location addressed by H-L pair.

Eg: MOV M, C (Move the contents of register to the memory loc^n whose address is in H-L pair;)

MVI r, data (Move immediate data to reg.)

[r] ← data ; states - 7 , flags - none

Addressing : Immediate , MC - 2

→ Here the 1st byte of the instruction is its opcode.
→ The 2nd byte of the instr^n is the data which is moved to reg. r.

Eg: MVI A, 05 (Move 05 to reg. A)

MVI M, data (Move immediate data to memory)

[H-L] ← data ; states - 10 ; flags - none

Addressing - Immediate / reg. indirect , MC - 3,

→ The data is moved to memory location whose address is in H-L pair.

Eg: LXI H, 2400H (Load H-L pair with 2400H)
MVI M, 08 (Move 08 to memory loc^n 2400H)
HLT (Halt)

LXI rp, data 16 (Load register pair immediate)

[rp] ← data 16 bits ; states - 10 , flags - none

Addressing - immediate , MC - 3 , [rh] ← 8 MSBs ; [rL] ← 8 LSBs of data

→ This instr^n loads 16-bit immediate data into reg. pair rp ; Only high order reg. is mentioned after the instruction.
→ Here, H in the instr^n stands for H-L pair, similarly LXI B is for B-C pair.

Eg: LXI H, 2500H (Loads 2500H into H-L pair)
→ Here 2500H denotes that the data 2500 is in hexadecimal.

LDA addr (Load Accumulator direct)

[A] ← [addr] ; states - 13 , flags - none

Addressing - direct , MC - 4.

→ The content of the memory location, whose address is specified by the 2nd and 3rd bytes of instr^n, is loaded into the accumulator.

Eg: LDA 2400H (Load the content of memory location 2400H into the Accumulator)

STA Addr (store ACC. direct)

[addr] ← [A], states - 13, flags - none
Addressing - direct, MC - 4.

→ The content of the acc. is stored the memory location whose addresses is specified in the 2nd and 3rd byte of instrn.

Eg: STA 2000H (store the content of the acc. in the memory loc^n 2000H)

LHLD addr (Load H-L pair direct)

[L] ← [addr], [H] ← [addr+1]., states - 16
flags - none, Addressing - direct, MC - 5

→ The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction, is loaded into reg. L. The content of reg. H is stored in the next memory loc^n.

Eg:
LHLD 2500H (Load the content of reg. L in the memory loc^n 2500H).

→ The content of reg. H is stored in the memory loc^n 2501H.

SHLD addr (store H-L pair direct)

[addr] ← [L], [addr+1] ← H, states - 16
flags - none, Addressing - direct, MC - 5.

→ The content of the memory loc^n, whose address is specified by the 2nd and 3rd bytes of the instr^n, is loaded into reg L. The content of reg. H is stored in the next memory loc^n.

Eg: SHLD 2500H (store the content of reg. L in the memory loc^n 2500H).

→ The content of reg. H is stored in the memory loc^n 2501H.

LDAX rp: (Load acc. indirect)

[A] ← [[rp]], states - 7, flags - none, Addressing - reg. indirect, MC - 2.

→ the content of the memory loc?, whose address is on the register pair rp, is loaded into the accumulator.

Eg:- LDAX B (Load the contents of the memory loc?, whose address on the B-C pair, into the accumulator)

→ This instruction is used only for B-C and D-E register pair.

STAX rp: (Store acc. indirect)

[[rp]] ← [A], states - 7, flags - none, Addressing - Reg. indirect, MC - 2.

→ The content of the acc. is stored on the memory loc? whose address is on the register pair, rp.

Eg:- STAX D (Store the contents of the acc. in the memory loc? whose address is on D-E pair)

→ This instr? is true only for reg. pairs B-C and D-E.

XCHG: (Exchange the contents of H-L with D-E pair)

[H-L] ↔ [D-E], states - 4, flags - none, Addressing - Register, MC - 1.

→ The contents of H-L pair are exchanged with contents of D-E pair.

(ii) Arithmatic Group

ADD R (Add register to acc.)

[A] ← [A] + [R] (The content of reg. R is added to the content of the acc., and the sum is placed in the acc.)

states - 4, Addressing - Register

flags - All; MC - 1

Eg: ADD B (The content of reg. B is added to the content of the acc. and sum is placed in the acc.)

ADD M (Add memory to acc.)

[A] ← [A] + [[H-L]] (The content of the memory loc. addressed by H-L pair and carry status are added to the content of the acc. The sum is placed in the acc.)

states - 7, Addressing - Reg. indirect

flags - all MC - 2.

Eg: LXI H, 8200H (Load H-L pair by 8200H)
ADD M (Add the content of 8200H by H-L pair and carry status are added to acc. & store in acc.)
HLT (Halt)

ADC R (Add register with carry to acc.)

[A] ← [A] + [R] + [CS] (The content of reg. R and carry status are added to the content of acc. & the sum is placed in acc.)

states - 4
flags - All

Addressing - Register
MC - 2.

Eg: ADC B (The content of reg. B and CS are added to the content of acc. & the sum is placed in acc.)

ADC M (Add memory with carry to Acc.)

[A] ← [A] + [[H-L]] [CS] (The content of the memory loc. addressed by H-L pair and CS are added to the content of acc. & sum is stored in acc.)

states - 7
flags - ALL

Addressing - Reg. indirect
MC - 2.

Eg:- LXI H, 2500H (Load H-L pair by 2500H)
ADC M (Add the content of 2500H and cs are added to the acc. & sum is stored in acc.)
SHLD 2502H (Store the content of reg. L in memory loc" 2502H & H in 2503H)
HLT (Halt)

ADI data (Add immediate data to acc.)
[A] ← [A] + data. (Add the immediate data to the content of acc)

states - 7
flags - All
Addressing - Immediate
MC - 2.

→ The 1st byte of the instr" is its opcode. The 2nd byte of the instr" is data, and it is added to content of the acc.. The sum is placed in the acc

Eg: ADI 08 (Add 08 to the content of the acc. and place the result in acc.)

ACI data (Add with carry immediate data to acc.)
[A] ← [A] + data + [CS]

Addressing - Immediate, states - 7, flags - All
MC - 2

→ The 2nd byte of the instr" (which is data) and the carry status are added to the content of the acc. The sum is placed in the acc.

Eg: ACI 09 (Add 09 with carry to the content of the acc. and place the result in acc.)

DAD rp (Add register pair to H-L pair)
[H-L] ← [H-L] + [rp]

Addressing - Register, states - 10, flags - CS
MC - 3

→ The contents of register pair rp are added to the contents of H-L pair and the result is placed in H-L pair. Only carry flag is affected.

Eg: DAD B (Add BC pair to H-L pair)

SUB r (Subtract reg. from acc.)
[A] ← [A] - [r] (The contents of reg. r is subtracted from the content of the acc., and the result is stored in acc.)

states - 4
flags - All
Addressing - Register
MC - 1.

Eg: SUB 08 (Subtract 08 from the contents of acc.)

(iii) Logical Group

The instructions of this group perform AND, OR, EXCLUSIVE - OR ... operations; compare, rotate or take complement of data on register or Memory.

ANA R (AND register with acc.)

[A] ← [A] ∧ [R] (The content of reg. R is ANDed with the content of the acc., and the result is placed in the acc.)

States - 4
Addressing - Register
flags - All
MC - 1

- All status flags are affected here.
- The flag cs is cleared, i.e. it is set to '0'
- Auxilliary carry flag AC is set to 1

Eg: ANA B (The content of reg. B is ANDed with the content of the acc., and the result is placed in the acc.)

ANA M (AND memory with accumulator)

[A] ← [A] ∧ [[H-L]] (The content of the memory location addressed by H-L pair is ANDed with the acc. The result is placed in the acc.)

States - 7
Addressing - Reg. indirect
flags - All
MC - 2.

All flags are affected
The cs flag is set to 0 and AC to 1

Eg: LXI H, 2700H (Load H-L pair by 2700H)
ANA M (The content of memory loc 2700H is ANDed with the acc. The result is placed in the acc.)
HLT (Halt)

ANI data (AND immediate data with acc.)

[A] ← [A] ∧ data (The 2nd byte of the instn is data which is ANDed with the content of the acc. The result is stored in the acc.)

States :- 7
Addressing - Register Immediate
MC - 2.
flags :- All

→ The cs flag is set to 0 and AC to 1.
Eg: ANI 08 (08 is ANDed with the content of acc. & the result is stored in the acc.)

ORA R (OR register with acc.)
[A] ← [A] V [R] (The content of reg. R is ORed with the content of the acc. & the result is stored in the acc.)

States - 4 Addressing - Register
flags - All MC - 1

→ Here all status flags are affected.
→ Carry and AC are cleared i.e. the CS and AC flags are set to 0.

Eg: ORA D. (The content of reg. D is ORed with the content of acc. & result is stored in the acc.)

ORA M (OR memory with acc.)
[A] ← [A] V [[H-L]] (The content of the memory location addressed by H-L pair is ORed with the contents of acc. & result is stored in the acc.)

States - 7 Addressing - Reg. Indirect
flags - All MC - 2

→ The result is placed in the acc.
→ The CS and AC flags are set to 0.

Eg: LXI H, 8200H (Load H-L pair by 8200H)
ORA M (The content of memory loc. 8200H is ORed with the contents of acc. and result is stored in acc.)
HLT (Halt)

ORI data (OR immediate data with acc.)
[A] ← [A] V data (Here data is the 2nd byte of instr. and it is ORed with the content of the acc. The result is placed in the acc.

States - 7 Addressing - Immediate
flags - All MC - 2

→ Here all status flags are affected.
→ The CS and AC flags are set 0.

Eg: ORI 08 (08 is ORed with the content of acc.)

XRA R (EXCLUSIVE-OR register with acc.)
[A] ← [A] V [R] (The content of reg. R is Exclusive-ORed with the content of the acc. & the result is stored in the acc.)

States - 4 Addressing - Register
flags - All MC - 1

→ Here all status flags are affected
→ The CS and AC flags are set to 0.

Eg: XRA D (The content of reg. D is EX-ORed with the content of acc. & result is stored in acc.)

iii CPI data (Compare immediate data with acc.)

[A] - data (The 2nd byte of the instr is data, and it is subtra-cted from the content of the acc. The status flags are set according to the result of subtraction. But the result is discarded.

OR

Co States - 7
flags - All
Addressing - Immediate
mc - 2

Al → The content of the acc. remains unchanged.

[Eg: CPI 02 (Compare 02 with the acc.)

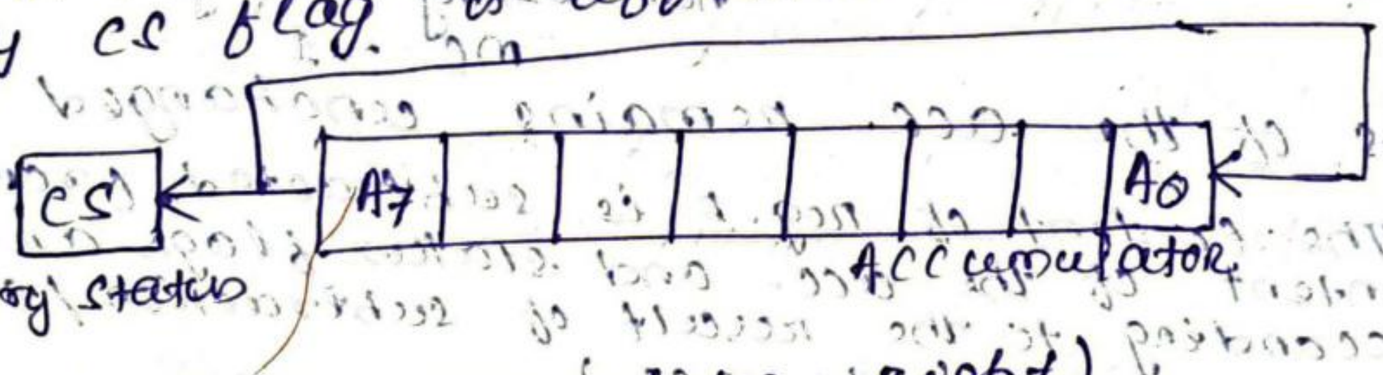
RLC (Rotate Accumulator left)

$[A_{n+1}] \leftarrow [A_n]$, $[A_0] \leftarrow [A_7]$, $[CS] \leftarrow [A_7]$

(The content of the acc. is rotated left by one bit. The seventh bit of the acc. is moved to carry bit as well as to the zero bit of the acc.)

states - 4
flags - CS
Addressing - Implicit
mc - 1

→ Only CS flag is affected here.



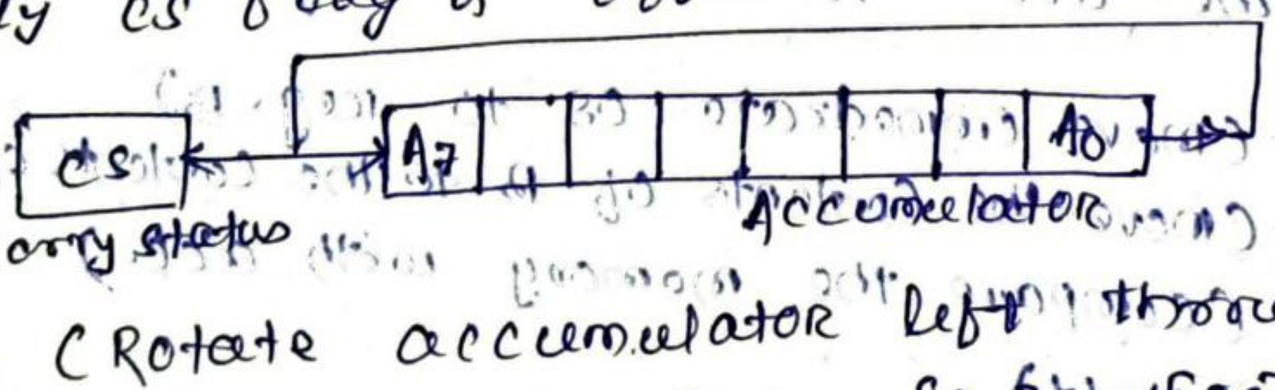
RRC (Rotate accumulator right)

$[A_7] \leftarrow [A_0]$, $[CS] \leftarrow [A_0]$, $[A_n] \leftarrow [A_{n+1}]$

(The content of the acc. is rotated right by one bit. The zero bit of the acc. is moved to the seventh bit as well as to carry bit.)

States - 4
flags - CS
Addressing - Implicit
mc - 1

→ Only CS flag is affected here.



RAL (Rotate accumulator left through carry)

$[A_{n+1}] \leftarrow [A_n]$, $[CS] \leftarrow [A_7]$, $[A_0] \leftarrow [CS]$

(The content of the acc. is rotated left one bit through carry. The seventh bit of the acc. is moved to carry and carry bit is moved to the zero bit of the acc.)

States - 4
flags - CS
Addressing - Implicit
mc - 1

→ Only carry flag is affected here.

(iv) Branch Group

The instructions of this group change the normal sequence of the program. There are two types of branch instructions. Conditional and unconditional.

→ The conditional branch instructions transfer the program to the specified level when certain condition is satisfied.

→ The unconditional branch instructions transfer the program to the specified label unconditionally.

JMP addr (label) (unconditional jump: jump to the instr specified by the address)

[PC] ← Label (the program jumps to the instr specified by the address (label) unconditionally)

states : 10
flags : None
Addressing : Immediate
mc : 3

→ 2nd byte and 3rd byte of the instruction give the address of the label where the program jumps.

→ The address of the label is the address of the memory loc for next instr to be executed.

Conditional jump addr (label)

→ After the execution of the conditional jump instr the program jumps to the instruction specified by the address (label) if the specified condition is fulfilled.

→ The program proceeds further in the normal sequence, if the specified condition is not fulfilled.

→ If the condition is true and program jumps to the specified label, the execution of a conditional jump takes 3 m/c, 10 states.

→ If condition is not true, only 2 m/c, 7 states are reqd for the execution of that instruction.

(i) JZ addr (label) (jump if the result is zero)

[PC] ← address (label), (the program jumps to the instr specified by the address (label) if the result is zero or jump if Z = 1)

states : 7/10
flags : None
Addressing : Immediate
mc : 2/3

→ Here the result of the preceding instr is taken into consideration

(ii) JNZ addr (label) (Jump if the result is not zero)

[PC] ← address (label), jump if Z=0

(The program jumps to the instruction specified by the address (label) if the result is non-zero (i.e the zero status Z=0))

states - 7/10,
flags - None

Addressing - Immediate
MC - 2/3

(iii) JC addr (label) (Jump if there is a carry)

[PC] ← address (label), jump if CS=1

(The program jumps to the instruction specified by the address (label) if there is a carry (i.e. the carry status CS=1))

states - 7/10
flags - None

Addressing - Immediate
MC - 2/3

→ Here the carry after the execution of the preceding instruction is under consideration

(iv) JNC addr (label) (Jump if there is no carry)

[PC] ← address (label), jump if CS=0

(The program jumps to the instruction specified by the address (label) if there is no carry (i.e the CS=0))

states - 7/10
flags - None

Addressing - Immediate
MC - 2/3

(v) JP addr (label) (Jump if the result is plus)

[PC] ← address (label) jump if S=0

(The program jumps to the instruction specified by the address (label) if the result is plus)

states - 7/10,
flags - None

Addressing - Immediate
MC - 2/3,

(vi) JM addr (label) (Jump if the result is minus)

[PC] ← address (label), jump if S=1

(The result is minus the program jumps upto the instruction specified by the address (label))

states - 7/10
flags - None

Addressing - Immediate
MC - 2/3

(vii) JPE addr (label) (Jump if even parity)

[PC] ← address (label), jump if even parity the parity status P=1.

(If the result contains even no. of 1's, the program jumps to the instruction specified by the address (label))

states - 7/10
flags - None

Addressing - Immediate
MC - 2/3

iii

(V) Stack, I/O and Machine Control Group

IN Port-address. (Input to accumulator from I/O Port)

[A] ← [Port]

States - 10, Addressing : Direct
flags - None MC : 3

→ The data available on the port is moved to accumulator
→ After instruction IN, the address of the port is specified.
→ The 2nd byte of the instruction contains the address of the port. The address of a port is an 8-bit addr

Eg: IN 01 (The address of the port B of an I/O port 8255.1 of a kit is 01)

OUT Port-address. (Output from acc. to I/O Port)

[Port] ← [A]

States - 10 Addressing : Direct
flags - None MC : 3

→ The content of the accumulator is moved to the port specified by its address.
→ After the OUT instruction, the port address is specified.
→ The 2nd byte of the instruction contains the address of the port.

Eg: OUT 00 (The address of the port A of an I/O port 8255.1 of a kit is 00)

PUSH rp: (Push the content of register pair to stack)

[[SP]-1] ← [rh]
[[SP]-2] ← [rl]
[SP] ← [SP]-2

States : 12 Addressing: Register (source)/reg. indirect (destination)
flags : None MC : 3

→ The content of the register pair rp is pushed into the stack.

PUSH PSW: (PUSH Processor status word)

[[SP]-1] ← A, [[SP]-2] ← PSW (Program status word), [SP] ← [SP]-2.

States : 12 Addressing: Reg. (source)/reg. indirect (destination)
flags : None MC : 3

→ The content of the acc. is pushed onto the stack.
→ The contents of status flags are also pushed into the stack.

→ The content of the register SP is decremented by 2 to indicate new stacktop.

POP rp. (Pop the content of reg. pair, which was saved, from the stack)

[rl] ← [[SP]], [rh] ← [[SP]+1], [SP] ← [SP]+2

states: 10
flags: None
Addressing: Reg. (destination) reg. indirect (Source)
MC: 3

→ The content of the register pair, which was saved earlier is moved from the stack to the register pair.

POP PSW. (POP processor status word)

PSW ← [[SP]], [A] ← [[SP]+1], [SP] ← [SP]+2.

states: 10
flags: All
Addressing: Reg. indirect
MC: 3

→ The processor status word which was saved earlier during the execution of the program is moved from the stack to PSW.
→ The content of the accumulator which was also saved is moved from the stack to the acc.

HLT. (Halt) (It is a m/c control operation instruction)

states: 5
flags: None
MC: 1.

→ The execution of the instruction HLT stops the program.
→ The registers and status flags remain unaffected.

XTHL. (Exchange stack-top with H-L)

[L] ↔ [[SP]], [H] ↔ [[SP]+1].

states: 16
flags: None
Addressing: Reg. indirect
MC: 5

→ The contents of the register L are exchanged with the byte of the stack-top.
→ The contents of the H register exchanged with the byte below the stack top.

SPHL. (Move the contents of H-L pair to stack pointer)

[H-L] → SP.

states: 6
flags: None
Addressing: Register
MC: 1.

→ The contents of H-L pair are transferred to the SP register.

4th chapter Branch & Sub-Routine Instruction

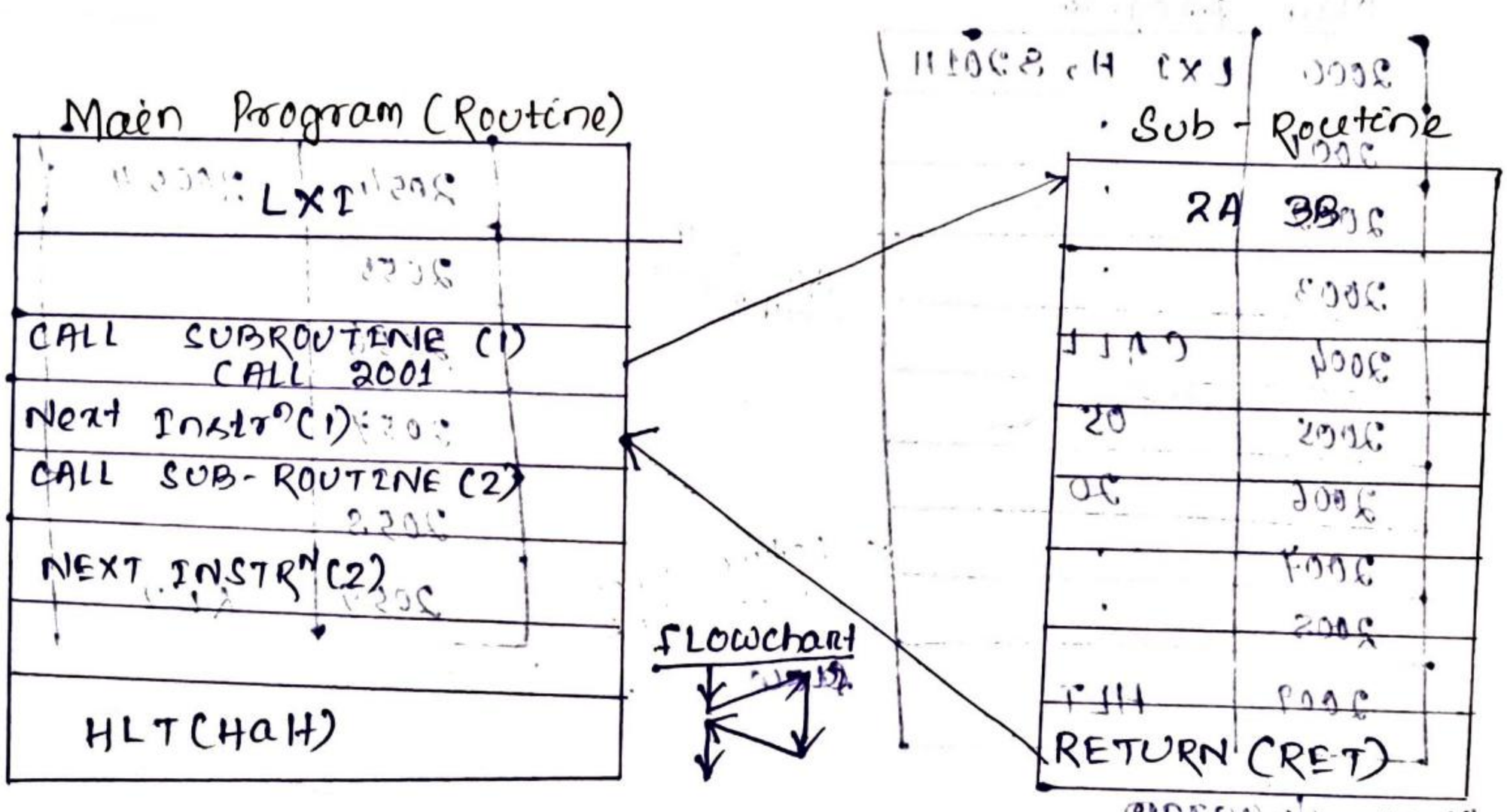
4.1 State & Explain Branching & Subroutine

Sub-Routine

The 8085 µp does not have the instruction for performing specific operations like multiplication, sine, cosine, square root, logarithm etc.

→ To perform the specific operation, one has to write a small program that may be stored in the memory.

→ The small program written for specific operation in the memory to be used inside the main program from specific operation is called subroutine.



→ A call instr^n is there to call the subroutine from the memory to the main program when it is req^d.

RET → RET instr^n is there to return the subroutine from the main program to the memory. One or more subroutine can be used in the main program.

subroutine saves the memory space.

Classification of subroutine →

- Sub routine
  - Simple Sub-routine
  - Nested Subroutine
  - Multiple sub-routine

iii)

Sample Subroutine

→ It is a set of instruction stored in the memory as a small program which is used by the single main program once or no. of times

→ To perform the specific operation, one has to write a small program that may be stored in the memory.

→ The small program written for specific operation in the memory to be used inside the main program for specific operation is called subroutine.

→ A call instr is there to CALL the subroutine from the memory to the main program when it is reqd.

Eg:-

Main program

2000	LXI H, 8201H
2001	.
2002	.
2003	.
2004	CALL
2005	05
2006	20
2007	.
2008	.
2009	HLT

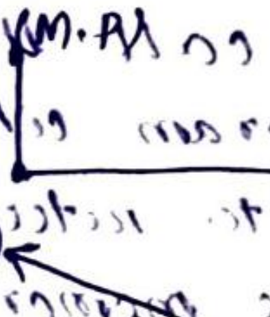
Go To the subroutine

Return from subroutine

Sub - Routine

2054	2005H
2055	
2056	
2057	
2058	
2059	RET

flow diagram



Nested Subroutine

→ This subroutine is formed by the transfer of control from one simple program which is located in a sub-routine which is already present under another sub-routine to the main program.

→ In this type of subroutine, there are two subroutines named subroutine-1 & subroutine-2.

→ A CALL instr" is there to CALL the subroutine-1 from the memory to the main program again another Call instruction is call the subroutine-2 from the subroutine-1
→ RET instr" is return from subroutine-2 to subroutine-1 & then from subroutine-1 to the main program.

Eg:-

Main Program

2000	LXI H,
2001	
2002	
2003	
2004	CALL
2005	
2006	
2007	
2008	
2009	
200A	
200B	

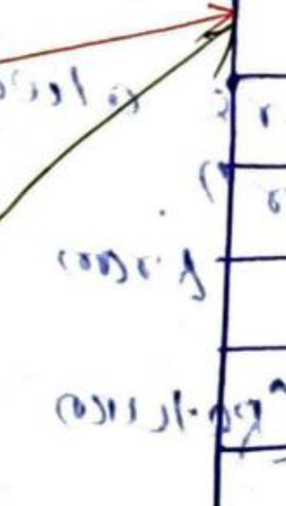
Sub-Routine-1

2054	2005 H
2055	
2056	CALL
2057	06
2058	20
2059	RET

Sub-Routine-2

3054	2056 H
	RET

FLOW Diagram



Multiple Subroutine

→ In this type of subroutine, One or main programs can use single subroutine by using CALL instr". After the execution of SR, the path will be returned from the SR to main program by using RET instr".

Main Program-1

2000	LXI
2001	
2002	CALL
2003	05
2004	20
2005	HLT

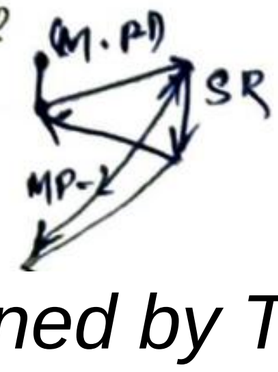
Main Program-2

3000	LXI,
3001	
3002	CALL
3003	
3004	
3005	HLT

Sub-Routine

2054	2005H
2055	
2056	
2057	
2058	
2059	
205A	RET

Flow Diagram



5th chapter ASSEMBLY LANGUAGE PROGRAMMING

A computer language is means of comm" betⁿ the user & the computer.

Machine Language →

machine language is a sequence of instr written in the form of binary nos consisting of 1's and 0's.

→ Binary coded instrⁿ is called machine code.

Advantage :-

→ faster in execution due to direct understandable for computer.
→ Computer hardware recognizes only machine cycle instrns.

Disadvantages :-

→ Difficult to understand the program.
→ Entry of program takes long time.
→ program is long
→ Writing the program is difficult.

Assembly Language →

Mnemonics → (It is a Greek word means mindful) The letter that suggests the operation to be performed in a particular instrⁿ is called "Mnemonics".

Eg :- ADD, MVI, MOV etc.

Assembly language programming (ALP) is a sequence of instrⁿ written in ~~mea~~ mnemonics to which computer responds indirectly.

→ It is not executed directly by the M/c.
→ An assembler is needed to translate assembly language program in the Object Code (M/c code).
→ An assembler is a system program software which is written by system programmer.
→ An assembler translate the assembly language program into M/c language program.

Examples of Assembly language Programs

1. place 05 in Register B

	M/m Addrs	M/C code	Mnemonics	Operands	Comments
Soln:	fC00	06, 05	MVI	B, 05	Get 05 in reg. B
	fC02	76	HLT		STOP.

2. Get 05 in register A; then move it to register B.

Soln:

fC00	MOV	A, 05	Get 05 in register A.
fC02	mov	B, A	Transfer 05 from register A to B
fC03	HLT		stop.

3. Load the Content of the memory Location fC50H directly to the accumulator, then transfer it to register B. The Content of the memory Location fC50 H to 05.

Soln:

fC00	LDA	fC50
fC03	mov	B, A
fC04	HLT	

4. Move the Content of the memory Location fC50H to register C. The Content of the memory Location fC50H is 08.

fC00	LXI	H, fC50
fC03	MOV	C, M
fC04	HLT	

5. Place the Content of the memory Location fC50H in reg. B and that of fC51H in reg. C. The contents of fC50 and fC51H are 11H and 12H respectively

fC00	LXI	H, fC50H
fC03	MOV	B, M
fC04	INX	H
fC05	MOV	C, M
fC06	HLT	

6. Place 05 in the accumulator. Increment it by one and store the result in the memory location fC50H

fC00	MVI	A, 05
fC02	INR	A
fC03	STA	fC50 H
fC06	HLT	

```
2009    HLT.
```

Data
2501 - 49H
2502 - 56H
Result
2503 - 9FH

12. Subtract 32H from 49H & 9BH from F8H

```
2000    LXI   H,2501H
2003    MOV   A,M
2004    INX   H
2005    SUB   M
2006    INX   H
2007    MOV   M,A
2008    HLT
```

Data
2501 - 49H
2502 - 32H
Result
2503 - 17H

Data
2501 - F8H
2502 - 9BH
Result
2503 - 5DH

13. Addition of TWO 8-bit Nos.; sum is 16-bits.

```
2000        LXI   H,2501H
2003        MVI   C,00
2005        MOV   A,M
2006        INX   H
2007        ADD   M
2008        JNC   AHEAD
200B        INR   C
200C  AHEAD STA   2503H
200F        MOV   A,C
2010        STA   2504H
2013        HLT
```

Data
2501 - 98H
2502 - 9AH
Result
2503 - 32H, LSBs of SUM
2504 - 01H, MSBs of SUM

Data
2501 - F5H
2502 - 8AH
Result
2503 - 7FH, LSBs of SUM
2504 - 01H, MSBs of SUM

14. Decimal Addition of two 8-bit nos, sum: 16 bits

```
2000        LXI   H, 2501H
2003        MVI   C, 00
2005        MOV   A, M
2006        INX   H
2007        ADD   M
2008        DAA
2009        JNC   AHEAD
200C        INR   C
200D AHEAD  STA   2503H
2010        MOV   A, C
2011        STA   2504H
2014        HLT
```

Data

2501 - 84D
2502 - 75D

Result

2503 - 59D, LSD
2504 - 01D, MSD

Data

2501 - 96D
2502 - 69D

Result

2503 - 65D, LSD
2504 - 01D, MSD

15. Addition of two 16-bit nos, sum: 16 bits or more

```
2000        LHLD  2501H
2003        XCHG
2004        LHLD  2503H
2007        MVI   C, 00
2009        DAD   D
200A        JNC   AHEAD
200D        INR   C
200E AHEAD  SHLD  2505H
2011        MOV   A, C
2012        STA   2507H
2015        HLT
```

Data

2501 - 98H
2502 - 5BH
2503 - 4CH
2504 - 8EH

Result

2505 - E4, LSB

Data

2501 - 45H
2502 - A6H
2503 - 23H
2504 - 9BH

Result

2505 - 68H

16. 8-Bit decimal Subtraction

```
2000    LXI  H, 2502H
2003    MVI  A, 99
2005    SUB  M
2006    INR  A
2007    DCX  H
2008    ADD  M
2009    DAA
200A    STA  2503H
200D    HLT
```

Data
2501 - 96
2502 - 38

Result
2503 - 58

Data
2501 - 99
2502 - 48

Result
2503 - 51

Data
2501 - 50
2502 - 10

Result
2503 - 40

17. Find 1's complement of an 8-bit no.

```
2000    LDA  2501H
2003    CMA
2004    STA  2502H
2007    HLT
```

Data
2501 - 96H

Result
2502 - 69H

Data
2501 - E4H

Result
2502 - 1BH

18. Find 1's complement of a 16-bit no.

```
2000    LXI  H, 2501H
2003    MOV  A, M
2004    CMA
2005    STA  2503H
2008    INX  H
2009    MOV  A, M
200A    CMA
200B    STA  2504H
200E    HLT
```

Data
2501 - B5H
2502 - 54H

Result
2503 - 7AH
2504 - ABH

Data
2501 - 75H
2502 - 89H

Result
2503 - 81H
2504 - 76H

19. find 2's complement of an 8-bit no.

```
2000    LDA   2501H
2003    CMA
2004    INR   A
2005    STA   2502H
2008    HLT
```

Data
2501 - 96H

Result
2502 - 6AH

Data
2501 - E4H

Result
2502 - 1CH

20. find 2's complement of a 16-bit no.

```
2000        LXI   H, 2501H
2003        MVI   B, 00
2005        MOV   A, M
2006        CMA
2007        ADI   01
2009        STA   2503H
200C        JNC   GO
200f        INR   B
2010   GO   INX   H
2011        MOV   A, M
2012        CMA
2013        ADD   B
2014        STA   2504H
2017        HLT
```

Data
2501 - 8C, LSB
2502 - 5B, MSB

Result
2503 - 74, LSB
2504 - A4, MSB

Data
2501 - 00
2502 - 5B

Result
2503 - 00
2504 - A5

21. Shift an 8-bit no. left by one bit

```
2000    LDA   2501H
2003    ADD   A
2004    STA   2502H
2007    HLT
```

Data
2501 - 65H

Result
2502 - CAH

28. To find larger of two nos.

```
2000        LXI  H,2501H
2003        MOV  A,M
2004        INX  H
2005        CMP  M
2006        JNC  AHEAD
2009        MOV  A,M
200A AHEAD  STA  2503H
200D        HLT .
```

Data

2501 - 98H
2502 - 87H

Result

2503 - 98H

Data

2501 - A9H
2502 - EBH

Result

2503 - EBH

29. To find the largest no. in a data array.

```
2000        LXI  H,2500H
2003        MOV  C,M
2004        INX  H
2005        MOV  A,M
2006        DCR  C
2007 LOOP   INX  H
2008        CMP  M
2009        JNC  AHEAD
200C        MOV  A,M
200D AHEAD  DCR  C
200E        JNZ  LOOP
2011        STA  2450H
2014        HLT .
```

Data

2500 - 03
2501 - 98
2502 - 75
2503 - 99

Result

2450 - 99

Data

2500 - 06
2501 - 38
2502 - 94
2503 - EB
2504 - A8
2505 - B5
2506 - FB

Result

2450 - FB