Von-Neuman Architecture	Hardvard Architecture.
→ It is a theoretical design based on the stored-program computer concept.	→ It is a modern computer architecture based on the Harvard Mark I relay-based comp model.
→ It uses same physical memory address for instructions and data.	→ It uses separate memory address for instructions and data.
→ Processor needs two clock cycles to execute an instruction.	→ Processor needs one cycle to complete an instruction.
→ Simpler control unit design and development of one is cheaper and faster.	→ Control unit for two buses is more complicated which adds to the development cost.
→ Data transfers and instruction fetches cannot be performed simultaneously.	→ Data transfers and instruction fetches can be performed at the same time.
→ Used in personal computers, laptops and workstations	→ Used in micro controllers and signal processing.

CHAPTER 2 : Introduction to 8085 μP

Features :

1. Introduced in 1976
2. 40-pin DIP (Dual Inline Package)
3. 8-bit μP
   - Data bus width = 8-bit
   - Address bus " = 16-bit
   - Total Addressable memory = $2^{16} = 65536$ byte = 64 KB
4. It uses single +5V power supply.
5. Operating frequency (max^m) = 5 MHz.
6. Upward compatible with 8080A
7. Supports large no. of instructions.

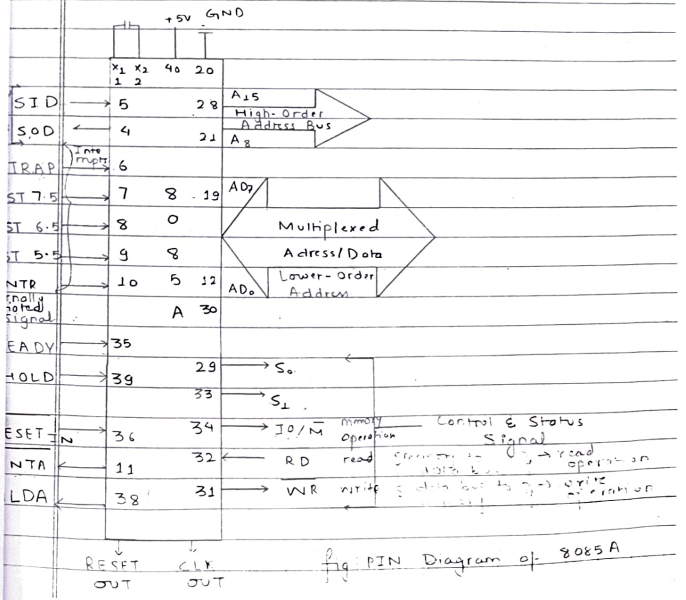
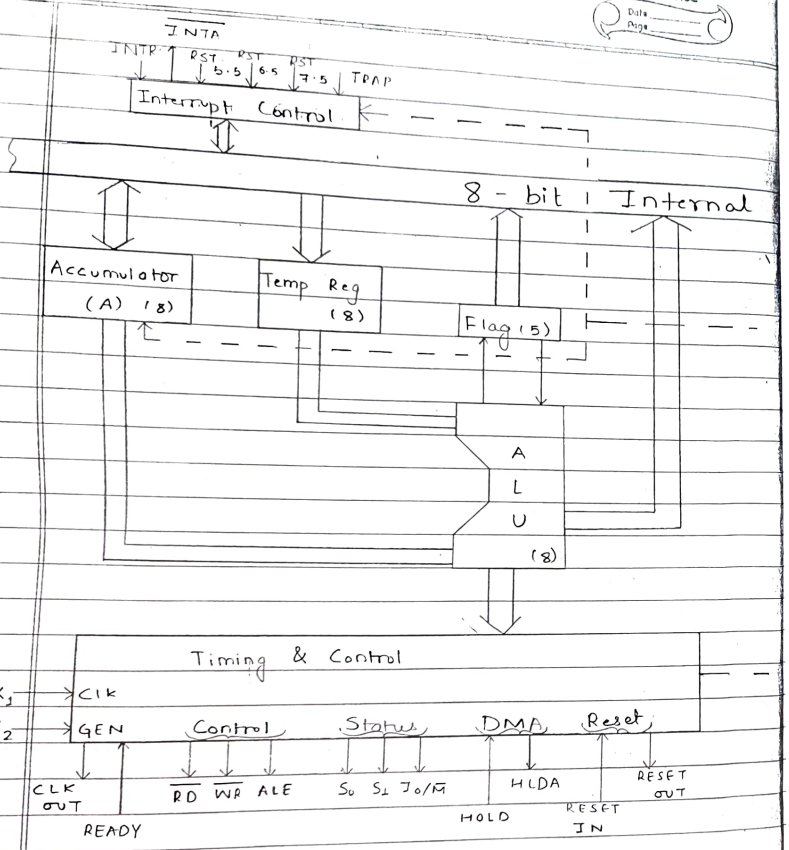


fig: PIN Diagram of 8085A



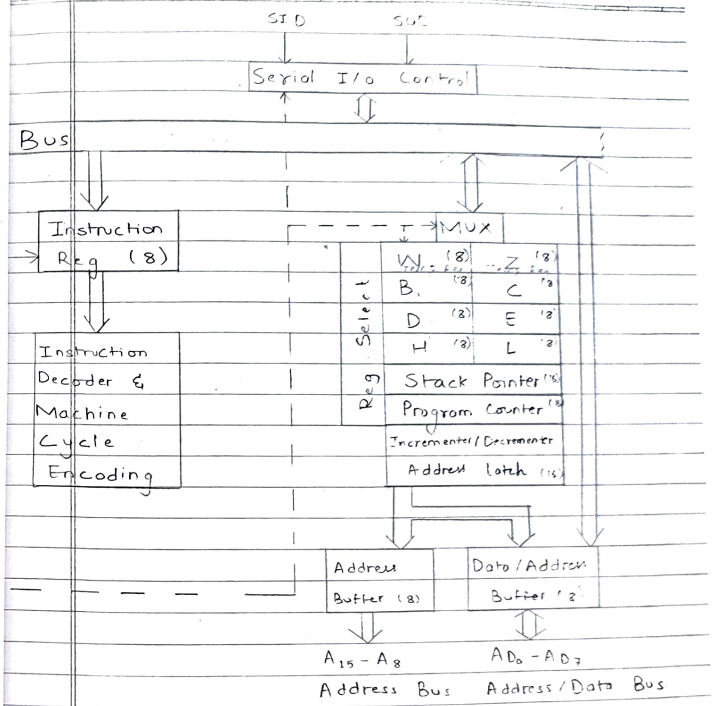


Fig Block diagram of 8085A

Flag Register of 8085

→ Flags are indicators that indicates the operation ongoing inside ALU.

→ Flag register are always associated with ALU.

→ There are five flags of 8085.

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

Fig Flag register of 8085

D_7 · 1 → set ; -ve → 1 → is set

1. Sign Flag → 0 → +ve
2. Zero Flag → 0 → 0 after branch 1
3. Auxillary Carry → Flag
4. Parity Flag → 0 → Even number of 1 ena bhane set = 1
5. Carry Flag → 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	1	0	1	0	1	0
1	1	0	0	0	1	0	1
1	0	0	1	1	0	1	0

8085 Instructions

→ An instruction is a command to the microprocessor to perform a given task.

→ A collection of instructions constitutes a program.

→ It consists of two parts:

1. Opcode (Operation code)

It is the first part of the instruction that represents the what kind of operation or task to be performed.

2. Operand:

Operands are the 2nd part of the instruction. This indicates the data to be operated in the instruction.

eg: MVI A, 25H

MVI A → Opcode

25H → Operand

Important

8085 Sets Instruction

Types of Instruction (~~According to byte size~~)

→ ~~Three categories of it~~

→ 8085 instructions can be grouped into five basic categories

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logic & bit manipulation Instructions
4. Branching Instruction
5. Machine control Instruction

1. Data Transfer Instructions:

- This instruction transfer or copies the instruction from one location (source) to another location (destination)
- These locations may be registers or memory location or I/O.
- Memory to memory transfer is not supported.
- Register pair (BC, DE, HL) is used for 16-bit data transfer.

Address: 0000H to FFFFH

Data (8-bit)

* Note: The I/O port or memory is address

V. V. Imp

	Instruction	Function	Example
1.	MVI Rd, data	This instruction immediately copies the 8-bit data into the register.	MVI A, 25H, MVI B, 3A H
2.	MOV Rd, Rs	This instruction copies the content of same Register into destination	MOV A, B, MOV B, H.
3.	LXI Rp, 16-bit data	This instruction immediately loads the 16-bit data into register pair (BC, DE and HL)	LXI B, 2019H, LXI H, 2076H
4.	IN 8-bit (port-addr)	This instruction copies the data byte from the port address specified in the instruction into Accumulator (A).	IN 03H. (8-bit data)
5.	OUT 8-bit (port_addr)	This instruction copies the data byte from the Accumulator into the port address specified in the instruction.	OUT 01H.
6.	STA 16-bit address (store Accumulator direct)	This instruction copies the data byte from the Accumulator into the memory location specified by 16-bit address in the Accumulator	STA 2050H (Address), A → 2050 (8-bit data)
7.	LDA 16-bit address (load Accumulator direct).	This copies the data type from the memory location specified by 16-bit address in the instruction into the Accumulator	LDA 2050 H, 2050 (LDA), 8-bit data (A)

	Instruction	Function	Example
8.	STAX Rp (store Accumulator Indirect)	This instruction copies the data byte from Accumulator into the memory locations specified by the register pair in the instruction	STAX B, STAX D, A → BC pair memory, 8-bit data
9.	LDAX Rp (Load Accumulator Indirect)	This instruction copies the data byte from the memory location specified by register pair in the instruction into the Accumulator.	LDAX B, LDAX D, data A ← DE pair 2050 H
10.	Mov M, R	Copies the content of specified register into the memory location specified by HL pair.	Mov M, B
11.	SHLD address	Store H & L Register Direct	SHLD 2050H
12.	LHLD address	Load H & L Register Direct	LHLD 2050H
13.	XCHG	Exchange the content of H & L with D & E.	

2. Arithmetic Instruction:

These instructions are capable of performing arithmetic operations including adding, subtraction, increment, decrement etc. The basic instructions are as follows.

	Instruction	Function	Example
1.	ADD R/M	• The content of Register / Memory is added to A and the result is stored in A.	ADD B A ← A+B
		• Memory (M) is specified by HL pair.	ADD M A ← A+M
2.	ADC R/M	• The content of R/M is added to A along with carry flag (CF) and result is stored in A.	ADC B A ← A+B+CF → 1 ADC M A ← A+M+CF
3.	ADI 8-bit data	• The 8-bit data is immediately added to A and the result is stored in A.	ADI 32H A ← A+32H
4.	ACI 8-bit data	• The 8-bit data is added to A along CF.	ACI 32H A ← A+32H+CF
5.	SUB R/M	• The content of R/M is subtracted from A and the result is stored in A.	SUB C A ← A-C
6.	SBB R/M	• The content of R/M is subtracted from A along with borrow & the result is stored in A.	SBB C A ← A-C-BF
7.	SUI 8-bit data	• The 8-bit data is subtracted from A and the result is stored in A	SUI 25H A ← A-25H
8.	INR R/M	• It increments the content of Register or Memory by 1	INR C C ← C+1

	Instruction	Function	Example
9.	DCR R/M	• It decrements the content of R/M by 1	DCR D D ← D-1
10.	INX Rp	• It increments the content of Register Pair by 1.	INX B BC ← BC+1
11.	DCX Rp	• It decrements the content of Register Pair by 1.	DCX D DE ← DE-1

3. Logic / Bit Manipulation Instruction

These instructions are capable of performing logical operations including logical multiplication (AND), logical addition (OR), XOR, complement (not), rotate compare, etc.

	Instruction	Function	Example
1.	ANA R/M	• The content of A are logically ANDed with the content of R/M and the result is stored in A.	ANA B A ← A·B ANA M
2.	ANI 8-bit data	• The content of A are logically ANDed with the immediate data specified in the instruction and the result is stored in A	ANI 25H
3.	ORA R/M	• OR	
4.	ORI 8-bit data	• OR	
5.	XRA R/M	• XOR	
6.	XRI 8-bit data	• XOR	

Qn. If A = 25H. What will be the result of execution of ANI 32H.

Ans;

		8 4 2 1
A = 25	=	0 0 1 0 0 1 0 1
32	=	0 0 1 1 0 0 1 0
AND		0 0 1 0 0 0 0 0
→ 20H		2 0

Qn. What are the status of flag in this execution.

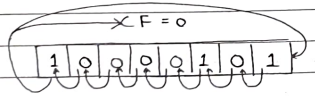
SF : Sign Flag → 0
CF : Carry Flag → 0
AF : Auxilliary Flag → 0
ZF : Zero Flag → 0
PF : Parity Flag → 0

7. Rotate Instruction

ⓐ RLC (Rotate Accumulator Left)
ⓑ RRC (Rotate " Right)
ⓒ RAL (" " Left with Carry)
ⓓ RAR (" " Right " ")

ⓐ RLC

For A = 85H



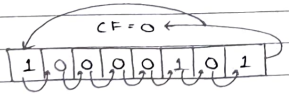
A = 0BH

CF = 1

0	0	0	0	1	0	1	1

ⓑ RRC

For A = 85H



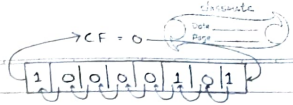
∴ A = C2H

CF = 1

1	1	0	0	0	0	1	0

ⓒ RAL

For A = 85H



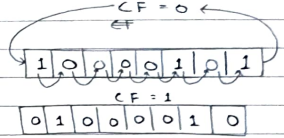
A = 0AH

CF = 1

0	0	0	0	1	0	1	0

ⓓ RAR

For A = 85H



8. Compare Instruction

ⓐ CMP R/M

This instruction compares the content of R/M with A. The result of comparison is:

If A < R/M : CF is set, ZF is reset
If A = R/M : ZF is set, CF is reset
If A > R/M : CF & ZF both are reset. Eg: CMP B

ⓑ CPI 8-bit data:

This instruction compares the immediate 8-bit data with A. The result of the comparison is:

If A < 8-bit data : CF is set, ZF is reset
If A = 8-bit data : ZF is set, CF is reset
If A > 8-bit data : CF and ZF both are reset. Eg: CPI 32H.

* Stack = temporary memory

4. Branching Instructions

These instructions are capable of transferring the program sequence from one location into another. It may be: I. Unconditional branch

	Instruction	Function	Example
1.	JMP 16-bit address	The program sequence is transferred to the memory location specified by 16-bit address	JMP C050H
2.	CALL 16-bit address	The program sequence is transferred to the subroutine program at the memory location specified by 16-bit address in the instruction.	CALL D050H
3.	RET	The program sequence is transferred from the subroutine program to the CALLing program	RET

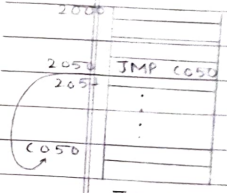
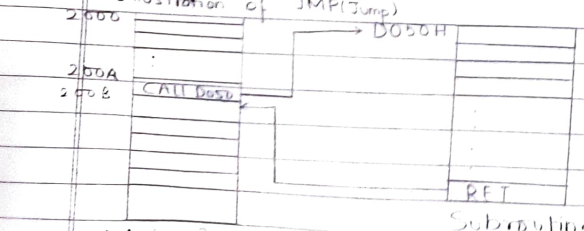


Illustration of JMP (Jump)



Main Program Subroutine Program

Illustration of CALL & RET

II. Conditional Branch

Instruction (Illustration)	Function	Example
JC 16-bit addr	Jump if Carry (CF → 1)	JC 2030H
JNC "	Jump if no-Carry (CF → 0)	JNC 2020H
JP "	Jump if positive (SF → 0)	
JM "	Jump if minus/negative (SF → 1)	
JZ "	" " zero (ZF → 1)	
JNZ "	" " not zero (ZF → 0)	
JPE "	" " parity even (PF → 1)	
JPO "	" " " odd (PF → 0)	

5. Machine Control Instructions

NOP → No operation is performed. eg NOP

HLT → The CPU finishes executing the current instruction & stops any further execution eg HLT

WAP to add two 8-bit numbers stored in memory location 2050H and 2060H respectively. Store the result in memory location 2070H.

→ Program

LDA 2050H	A ← Data1	2050 Data1
MOV B, A	B ← Data1	2060 Data2
LDA 2060H	A ← Data2	2070 Sum
ADD B	A ← A+B	
STA 2070H		
HLT		

WAP to add two 8-bit numbers stored in memory location 2050H and 2060H respectively. Store the sum in memory location 2070H & carry (if occurs) in 2071H.

2050	Data 1
2060	Data 2
2070	Sum
2071	Carry

→ Program

```
LDA  2050H      A ← Data 1
MOV  B, A       B ← Data 1
MVI  C, 00H
LDA  2050H
MOV  B, A
LDA  2060H
ADD  B          A ← A + B
JNC  loop
INR  C
loop: STA  2070H
MOV  A, C
STA  2071H
HLT
```

WAP to multiply two immediate numbers. And store the result in C050H.

→ Program

```
MVI  A, 00H
MVI  B, 05H
MVI  C, 03H
loop: ADD  B,   A ← 5+0+5+5
DCR  C          C = 2, 1, 0
JNZ  loop
STA  C050H
HLT
```


Types of Instruction (on the basis of size)

① One byte - Instruction:

These instruction occupy a total memory of one byte. It includes the opcode & operand in the same byte. eg: ADD B, MOV A, B, RAL, ANA B, etc.

② Two-byte Instruction:

These instructions occupy a total memory of two bytes. The first byte specifies the opcode and the second byte specifies the operand.

eg: MVI A, 32H, ADI 25H, IN 30H, etc.

③ Three byte Instructions:

These instruction occupy a total memory of three bytes. The first byte specifies the opcode and the remaining two bytes specifies the operand.

eg: STA 2050H, JNC 3050H, LXI H, 2070H

Imp

Addressing Modes of 8085:

- An instruction consits of opcode and operand. The way in which operands are expressed within an instruction is called addressing modes.
- Addressing modes explain how operands are addressed within an instruction.
- There are five addressing mode of 8085.

① Immediate Addressing Mode:

If the 8-bit or 16-bit operands/data are immediately expressed within an instruction then it is called immediate addressing mode.

Eg: MVI A, 29H, LXI B, 2050H, ADI 32, ANI FFH

② Register Addressing Mode:

If the operands are stored in register and the operation is in betn register that are expressed in the instruction then it is called Register Addressing Mode. Eg: MOV A, B, ADD B, ANA D.

③ Direct Addressing Mode:

If the operand's effective address (address where the operand is stored) is expressed with in an instruction then it is called direct addressing mode.

eg: STA 2050, LDA 3050H, SHLD 2050, IN 01H, OUT 03H.

④ Indirect Addressing Mode:

If the register pair which contain the address of the data is specified within an instruction, then it is called indirect addressing mode. Here, register pair hold the ~~operation~~ memory location. Eg: STAX D, LDAX B, MOV A, M.

⑤ Implied Addressing Mode:

If the opcode in an instruction tells about the operand & the operand is not visible within an instruction then it is called implied addressing mode.

Eg: HLT, RET, NOP, RAL, RRC, etc.

Timing Diagram of 8085 Instructions:

Related Terms:

1 Instruction Cycle: The time taken to complete the execution of an instruction is called instruction cycle. It is the combination of machine cycles

2 Machine Cycle: The time taken by the processor to access memory location, I/O ports, or to acknowledge an interrupt once is called machine cycle. It is combn of T-states.

3 T-state: It is the subdivision of operation performed in one clock cycle of the processor's clock.

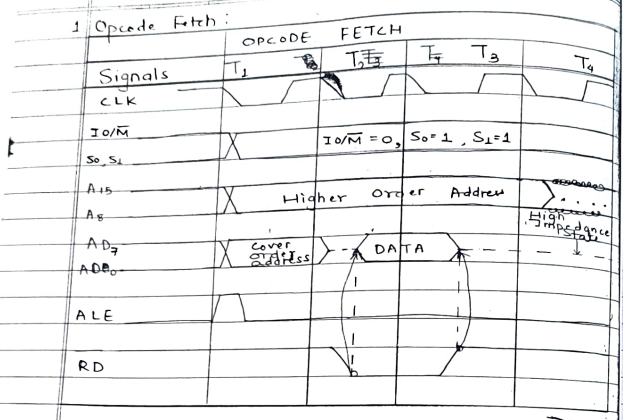
Machine Cycle & the Status Single

Machine cycle	Status Signal			T-states
	$IO/\bar{M}$	S_0	S_1	
Opcode Fetch	0	1	1	4T/6T
Memory Read	0	0	1	3T
Memory Writ	0	1	0	3T
I/O Read	1	0	1	3T
I/O Write	1	1	0	3T

Other machine cycles may be Interrupt Acknowledge & Bus Idle.

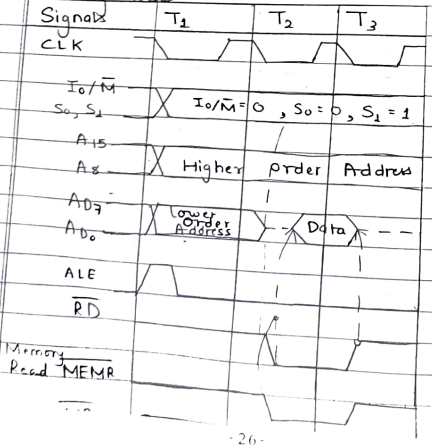
Note: Timing diagram is a graphical representation of an instruction cycle that consists of fetch, decode and execute cycle.

1. Opcode Fetch:



State change @ falling edge of clock cycle.

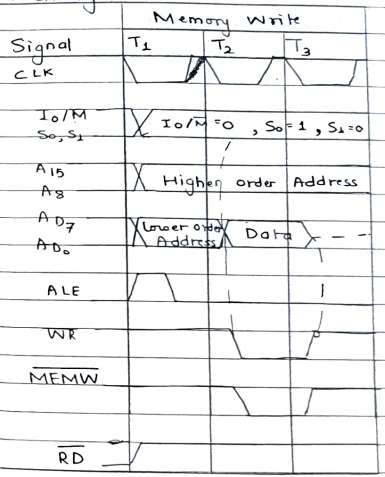
2. Memory Read



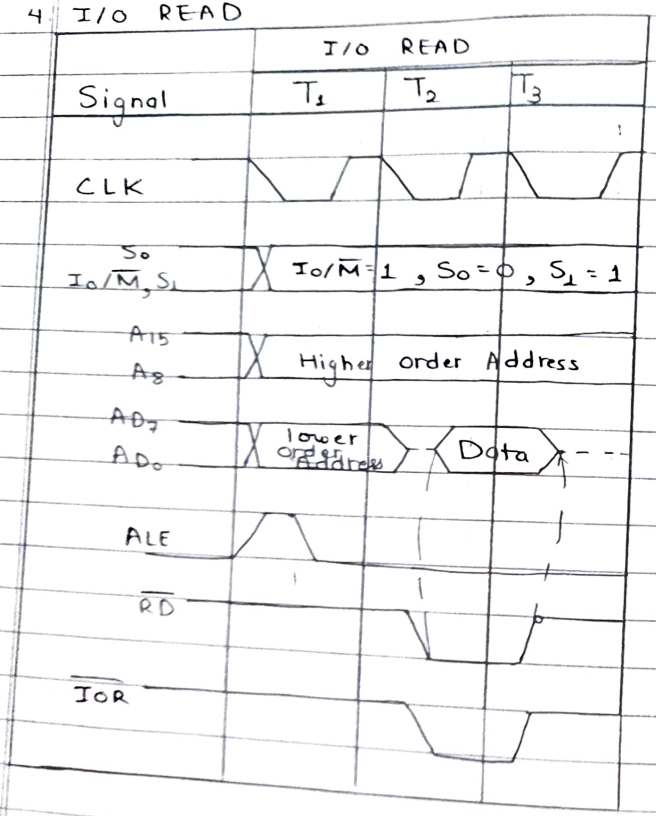
Signals	T1	T2	T3
WR			

→ Program counter placed higher order bus at higher bus and lower order bus at lower bus at time.

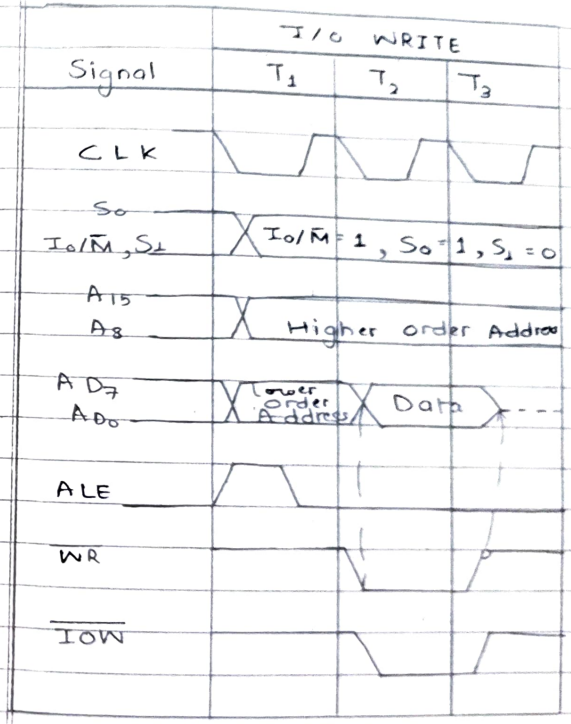
3. Memory Write:



4 I/O READ



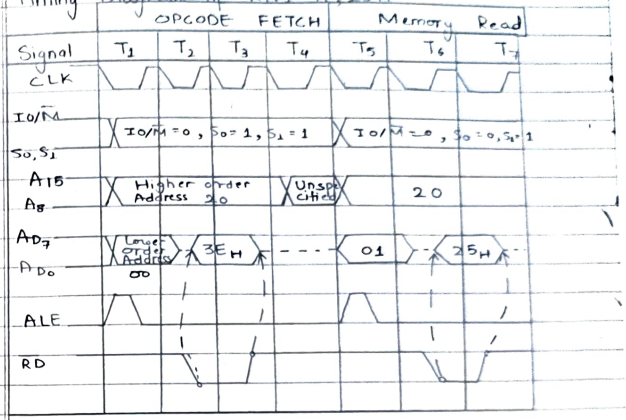
5 I/O WRITE



List of Instructions (along with machine cycles)

	Instruction	Size	Machine Cycle
1	MOV A, B	1 byte	1 Opcode Fetch
2	MVI A, 32H	2 byte	1. Opcode Fetch 2. Memory Read (MR)
3	MVI M, 32H	2 byte	1. Opcode Fetch 2. MR 3. MW
4	STA 2065H LDA 2065H	3 byte	1. Opcode Fetch 1 OF 2. MR 2 MR 3. MR 3 MR 4. MW 4 MR
5	MOV A, M	1 byte	1. Opcode Fetch 2. MR
6	MOV M, A	1 byte	1. OF 2. MW
7	IN 84H	2 byte	1. OF 2. MR 3. I/O Read
8	OUT 01H	2 byte	1. OF 2. MR 3. I/O Write
9	ADI 12H	2 byte	1. OF 2. MR
10	JNZ C050H	3 byte	1. OF 2. MR 3. MR (If condition is true)
11	STAX B	1 byte	1. OF 2 MW

Timing Diagram of MVI A, 25H



The above timing diagram can be explained as

1 This instruction consists of two bytes. The first is opcode & the second is data byte. The MP need these bytes to be read from the memory and thus requires two machine cycles,

I Opcode Fetch
II Memory Read

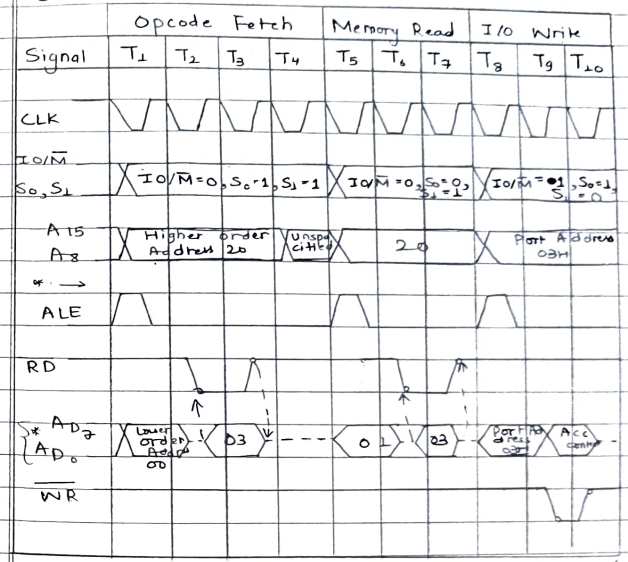
2 At T_1, the MP identifies that it is a opcode fetch cycle. It places $IO/\overline{M} = 0$, $S_0 = 1$, $S_1 = 1$. It places the memory address (say 2000H) from the program counter on the address bus A_{15}-A_8 and AD_7-AD_0 and increments the program counter to next memory address 2001H to point to the next machine code. The ALE goes high during T_1. The 8085 asserts the $\overline{RD}$

control signal which enables the memory and memory places the byte 3EH from location 2000H on the data bus. The 8085 places the opcode in the instruction register & disables the $\overline{RD}$ signal. The fetch cycle is completed in state T_3. During T_4, 8085 decodes the opcode and finds out that a second byte needs to be read. After T_3 state, the content of $A_{15} - A_8$ are unknown and the data bus $AD_7 - AD_0$ goes into high impedance state.

3. After completion of opcode fetch cycle, the MP places the address 2001H on the address bus and increments the program counter at T_5. The second machine cycle is identified as memory Read cycle according to the control signal generated i.e $IO/\overline{M} = 0$, $S_0 = 0$, $S_1 = 1$. The ALE goes high during T_5 state. At T_6, the $\overline{RD}$ signal becomes active and enable the memory chip.

4. Then the memory places the data byte 25H on the data bus, and the 8085 reads and store the byte in Accumulator during T_7.

Q.n Draw and explain the timing diagram for OUT 03H.



OUT 03H

2000 → D3

2001 → 03

Assembly Language Program

Qn WAP to find the sum of 5 data bytes stored from memory 2000H and store the sum in memory location 2005H.

→
```
LXI H, 2000H
MVI C, 05H
```

2000 → Data 1
⋮
2004 → Data 5
2005 → Sum.

```
    MVI A, 00H
    LXI H, 2000H
    MVI C, 05H
loop: ADD M
    DCR C
    JNZ loop
    STA
```
↳ INX H

```
MVI A, 00H
LXI H, 2000H
MVI C, 05H
loop: ADD M
    INX H
    DCR C
    JNZ loop
    STA 2005
    HLT
```

Qn To transfer a block of 10 bytes of data stored from memory 2000H into another block of memory starting from 3000H.

→
```
LXI H, 2000H
LXI D, 3000
MVI C, 0AH
loop: MOV A, M
```

```
STAX  D
INX   H
INX   D
DCR   C
JNZ   loop
HLT
```