LECTURE NOTES

ON

MICROPROCESSOR & MICROCONTROLLER

(INTEL-8085)

Prepared by

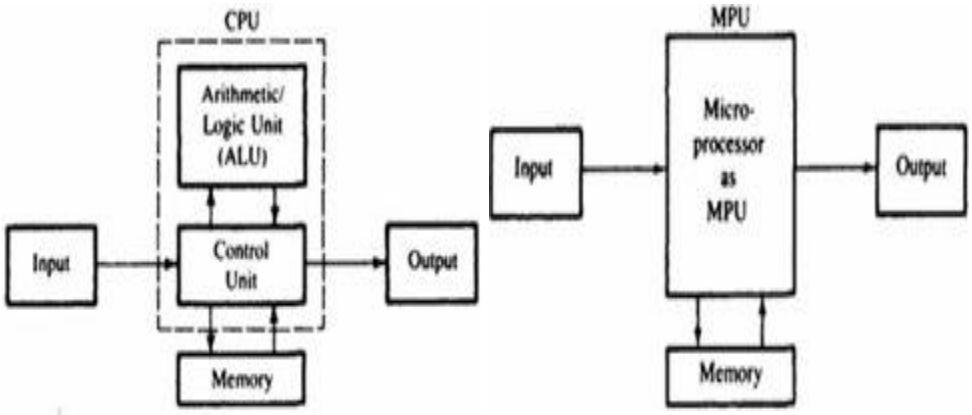
DR. MINU SAMANTRAY

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

TRIDENT ACADEMY OF TECHNOLOGY, BHUBANESHWAR

What is Microprocessor?

- A microprocessor is a clock-driven semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale integration (LSI) or very-large-scale integration (VLSI) technique.
- The microprocessor is capable of performing various computing functions and making decisions to change the sequence of program execution.
- In large computers, a CPU performs these computing functions. The Microprocessor resembles a CPU exactly.
- The microprocessor is in many ways similar to the CPU, but includes all the logic circuitry including the control unit, on one chip.
- The microprocessor can be divided into three segments for the sake of clarity. – They are: arithmetic/logic unit (ALU), register array, and control unit.
- A comparison between a microprocessor, and a computer is shown below:



- **Arithmetic/Logic Unit:** This is the area of the microprocessor where various computing functions are performed on data. The ALU unit performs such arithmetic operations as addition and subtraction, and such logic operations as AND, OR, and exclusive OR.
- **Register Array:** This area of the microprocessor consists of various registers identified by letters such as B, C, D, E, H, and L. These registers are primarily used to store data temporarily during the execution of a program and are accessible to the user through instructions.

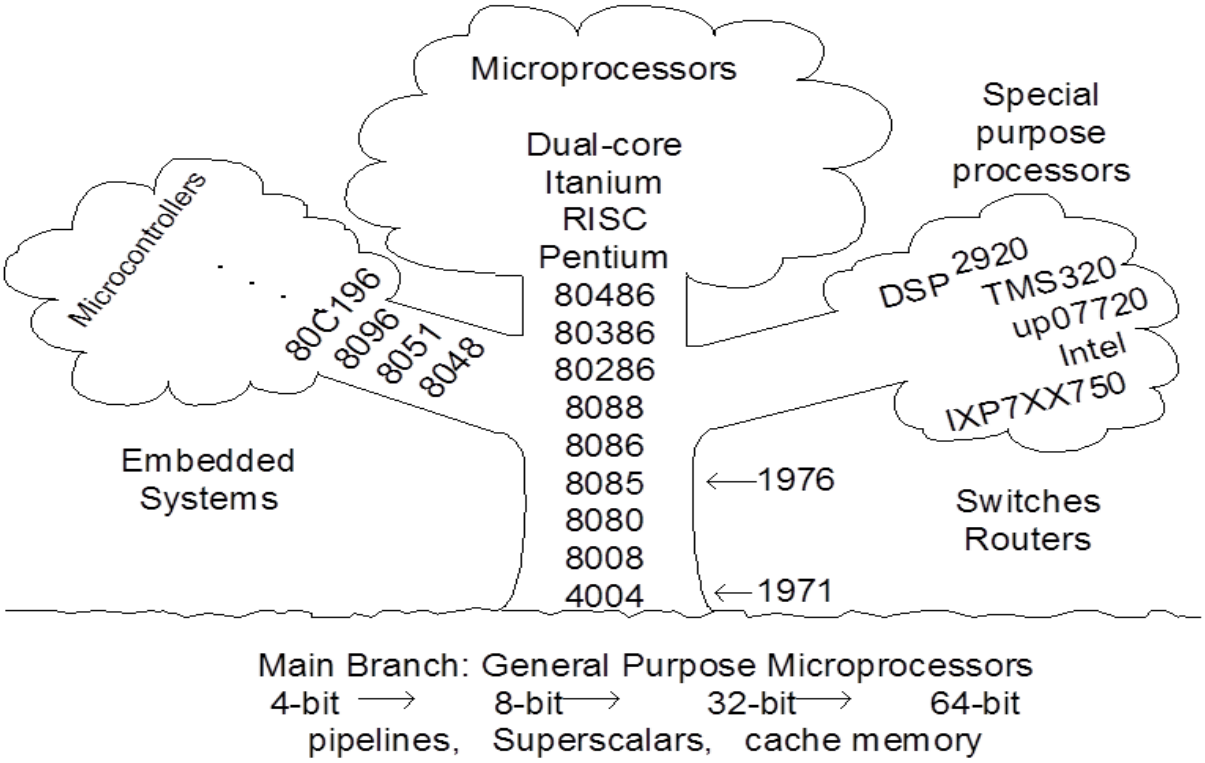
- **Control Unit:** The control unit provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor and memory and peripherals.

- **Memory:** Memory stores such binary information as instructions and data, and provides that information to the microprocessor whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Results are either transferred to the output section for display or stored in memory for later use. Read-Only memory (ROM) and Read/Write memory (R/WM), popularly known as Random- Access memory (RAM).

1. The ROM is used to store programs that do not need alterations. The monitor program of a single-board microcomputer is generally stored in the ROM. This program interprets the information entered through a keyboard and provides equivalent binary digits to the microprocessor. Programs stored in the ROM can only be read; they cannot be altered.
2. The Read/Write memory (RIWM) or Random Access Memory (RAM) is also known as user memory. It is used to store user programs and data. In single-board microcomputers, the monitor program monitors the Hex keys and stores those instructions and data in the R/W memory. The information stored in this memory can be easily read and altered.

- **I/O (Input/Output):** It communicates with the outside world. I/O includes two types of devices: input and output; these I/O devices are also known as peripherals.
- **System Bus:** The system bus is a communication path between the microprocessor and peripherals: it is nothing but a group of wires to carry bits.

Evolution of microprocessor



Application of microprocessor

Microprocessor is a multi-use device which finds applications in almost all the fields. Here is some sample applications given in variety of fields.

Electronics:

- Digital clocks & Watches
- Mobile phones
- Measuring Meters

Mechanical:

- Automobiles
- Lathes
- All remote machines

Electrical:

- Motors
- Lighting controls
- Power stations

Medical:

- Patient monitoring
- Most of the Medical equipments
- Data loggers

Computer:

- All computer accessories
- Laptops & Modems
- Scanners & Printers

Domestic:

- Microwave Ovens
- Television/CD/DVD players
- Washing Machines

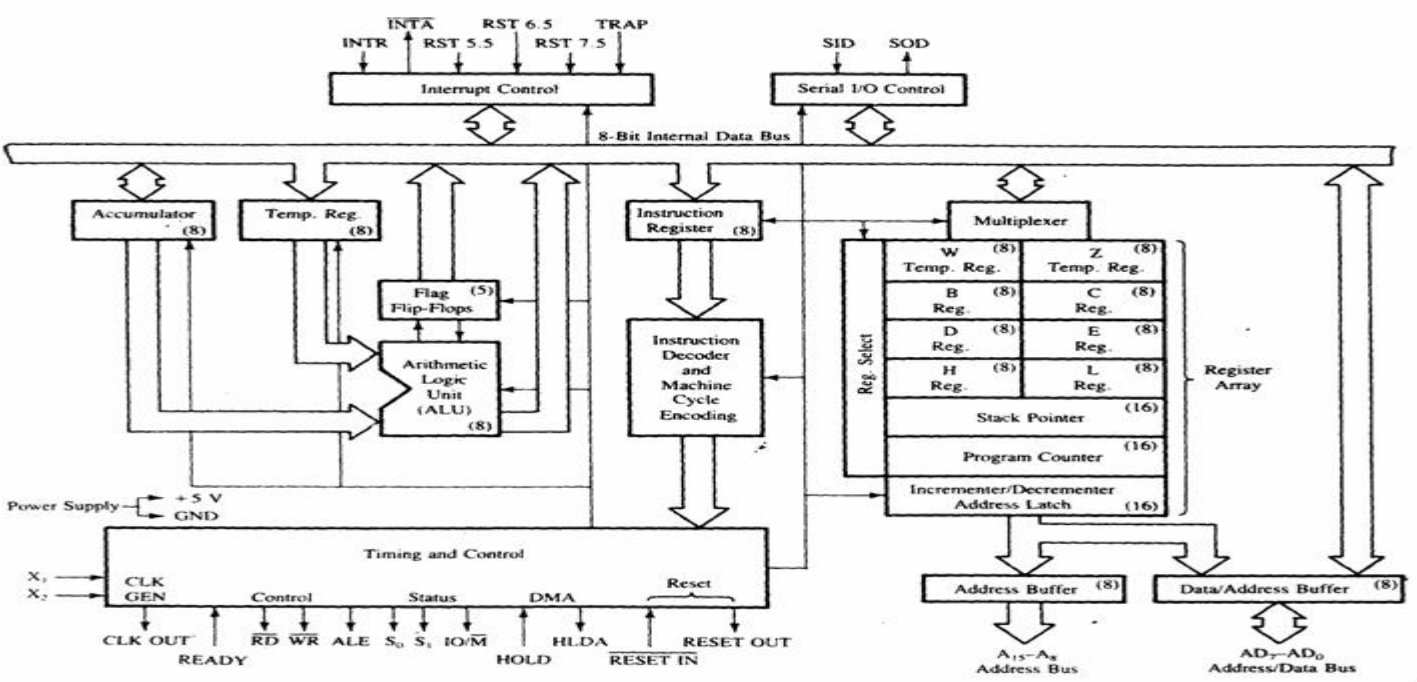
INTEL-8085

FEATURES

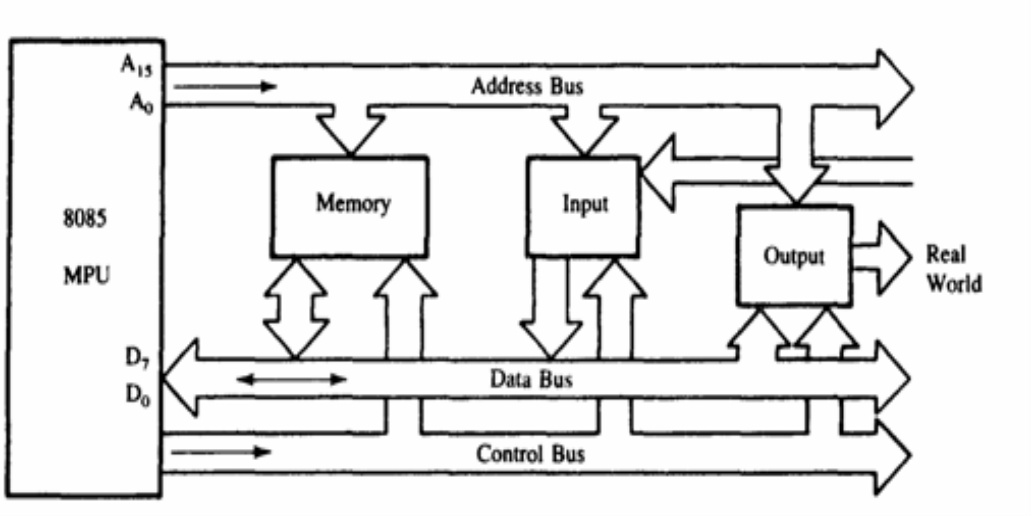
- It is an 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 2^{16} = 65536 bytes (64KB) memory locations through A_0-$A_{15.}$
- The first 8 of address lines & 8 data lines are multiplexed $AD_0 – AD_7$.
- Data bus is a group of 8 lines $D_0 – D_7$.
- It supports external interrupt request.
- 16 bit programs counter (PC).
- 16 bit stack pointer (SP).
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

ARCHITECTURE

The functional block diagram or architecture of 8085 Microprocessor is very important as it gives the complete details about a Microprocessor. Fig. shows the Block diagram of a microprocessor.



BUS ORGANISATION



(Bus organisation of 8085)

ADDRESS BUS

- Length is 16 bits generally identified as A_0 to A_{15}.
- Unidirectional i.e. bit flow in one direction.
- Address information flows from MP to Peripherals or memory.
- MP uses the address bus to identify the peripherals or memory locations.

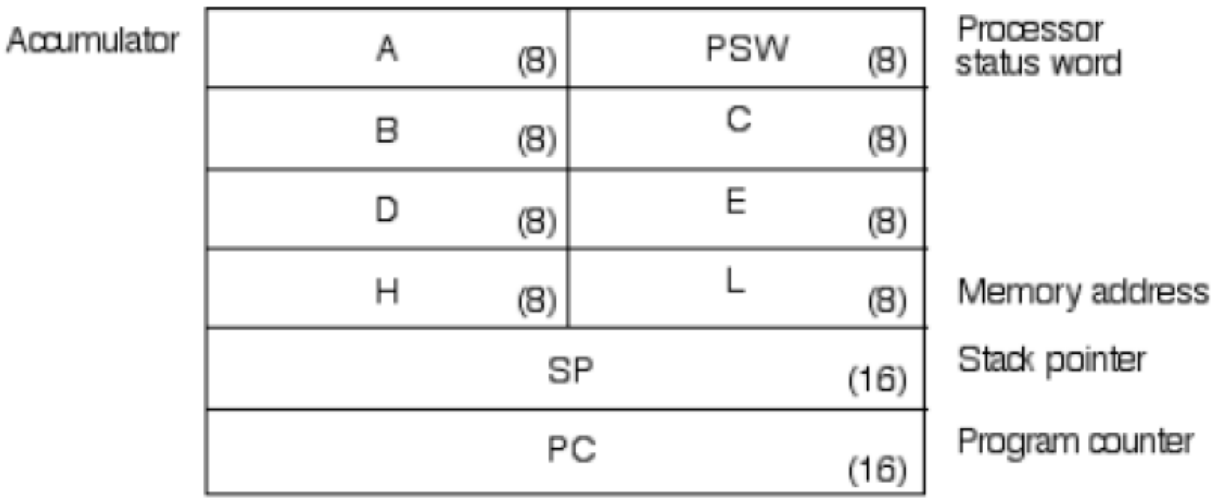
DATA BUS

- Length is 8bits generally identified as D_0 to D_8.
- Bidirectional i.e. data flow in both directions between MP and peripherals.
- MP uses the Data bus for transferring the data.
- Data bus determines the word length and register size of a MP.

CONTROL BUS

- The definition of bus is not satisfied in the case of control bus because it is not the group of wires or connecting paths rather it comprised various single wire or line to carry synchronization signal.
- The control signal flow in the form of pulse, either low or high to indicate the various operations.
- The control signals are MEMR* & IOR* when MP performing a READ operation and MEMW* & IOW* when MP performing a WRITE operation.

REGISTER ORGANISATION



There are two types of register available in 8085. They are

- GPR (general purpose register)
- SFR (Special function register)

GPRs

- There are six general purpose registers available in 8085. They are B, C, D, E, H, L and the size of all the registers are 8 bits i.e. they can store 8 bits of binary information temporarily during program execution.

- But they can be used as pairs for 16 bit information like BC, DE, and HL. Among them B, D, H are the higher order registers and C, E, L are the lower order registers.

- From above three pairs only HL register pair is used for accessing the memory.

SFRs

ACCUMULATOR

- This register is considered as the part of ALU (Arithmetic logic Unit). That means for any arithmetic and logical operation one of the operand is must store in the accumulator.

- Most importantly after the A/L operation the result is stored in this register. For that purpose the name is called accumulator. Also it is used to store the 8 bits of data.

PROGRAM COUNTER

- It is a 16 bits of register used to store the address of the next instruction to be fetched.
- The content of this register is automatically incremented by one after each byte is fetched. It is otherwise called as memory pointer.
- Because of this register the microprocessor executes all the instructions sequentially.

STACK POINTER

STACK

- A stack is the group of memory location defined in the R/W memory (RAM) which is used to store the data temporarily during program execution.
- It is always defined by the user. Stack is always works on the principle of LIFO (last in fast out). That means the data first enter will retrieve last. So in the stack data can store by decrementing the address and can retrieve by incrementing the address.
- Stack is always accessed in 16 bit operation.

STACK POINTER

- The stack pointer is the 16 bits register which store the starting address of the stack.
- So data stored into the stack by decrementing the content of the SP and retrieve by incrementing the content of the SP.
- It always advisable to the users to defined the stack in the higher end of the memory to avoid the overlapping between the main program and stack.
- The SP is initialized by the instruction LXI SP, 16bits address. The stack can access by using the two instructions like PUSH and POP.

FLAGS

- This is the 8 bits register and used to reflect the condition of the result of any A/L operation.
- Among 8 flip flops only 5 flip flops are used.
- Due to this flags MP take the decision to change the sequence of program execution i.e. it can jump from one location in the memory to another during program execution.
- This flags depend on the content of the accumulator because after each A/L operation the result is stored in the accumulator.

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

Flag register

- The five flags are
  1. Carry flag
  2. Parity flag
  3. Auxiliary carry flag
  4. Zero flag
  5. Sign flag

CARRY FLAG

After each A/L operation if there will be a carry result then this flag is set otherwise it is reset. This acts as a borrow flag in the case of subtraction operation. For example when the two 8bits numbers like 80H (1000000) and 82H (10000010) when added the result will be 102H (1, 00000010), so there is a carry results and in this case the CF will be set.

PARITY FLAG

If the nos. of 1's present is even in the result (content of accumulator) after any A/L operation then this flag is set otherwise it will be reset. For example if the accumulator contains a data like F2H (11110010) after an A/L operation, then PF will be set because the result contains six nos. of 1's i.e. even nos. of 1's.

ZERO FLAG

- If the accumulator contains 00H (0000000) after any A/L operation then this flag is set otherwise reset. For example if the two same numbers is subtracted then the result will be 00H, then in this case the ZF is set.
- It is not only depends the accumulator but also depends the other GPRs (B,C, used as counter) when INR and DCR instructions is executed.

SIGN FLAG

- If the MSB (D7 bit) in the accumulator is 1 after any A/L operation then this flag will be set otherwise it will reset.
- This flag is exclusively used in sign operation. As we know the MSB bit is used to represent the sign of a number i.e if the MSB of a number is 1 then that number is a negative number so this flag depend on the MSB bit of the accumulator.

- In sign number manipulation among 8 bits only 7 bits are used for the magnitude and last bit is used for sign.

AUXILLARY FLAG

- This flag is not accessible to the user and no instruction is available using this flag. This is used exclusively in BCD operation.
- In any arithmetic operation when there will be a carry result from D3-bit to D4-bit of the operand then this flag is set otherwise it is reset.

INSTRUCTION

- An instruction is a binary pattern which is used to command the MP to perform a given task on specified data.
- Each instruction has two parts: op-code (operational code) and operand. Op-code specifies the type of operation to be performed and operand specifies on which data that operation is to be performed.

The operand is specified in the instruction in various ways. It may be the 8/16 bits data or address or the content of memory location or the content of register or it is defined implicitly.

INSTRUCTON WORD SIZE

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

1. 1-byte instructions
2. 2-byte instructions
3. 3-byte instructions

ONE-BYTE INSTRUCTIONS

A 1-byte instruction includes the op-code and operand in the same byte i.e. this instruction required one memory location to be stored.

EXAMPLES

Op-code	Operand	Binary Code	Hex Code
MOV	C, A	0100 1111	4FH
ADD	B	1000 0000	80H
CMA		0100 1111	2FH

TWO-BYTE INSTRUCTION

In this instruction first byte specifies the op-code and the second byte specifies the operand but in some instruction first byte not only specifies the opcode part but also some portion of the operand (e.g MVI Reg, 8-bit). It required two memory locations to be stored. In general 8-bits data /address attached in the instruction.

EXAMPLES

Op-code	Operand	Binary Code	Hex Code
MVI	A, 32H	0011 1110	3EH
		0011 0010	32H
ADI	F2H	1100 0110	C6H
		1111 0010	F2H

THREE BYTE INSTRUCTIONS

In this type, the first byte specifies the op-code part and the next two byte specifies the operand part. In the lower address of memory the second byte (LSB 8-bits) is stored and in the higher address the third byte (MSB 8-bits) is stored. These instructions are required three locations to store in the memory.

EXAMPLES

Op-code	Operand	Binary Code	Hex Code
LDA	2050H	0011 1010	3AH
		0101 0000	50H
		0010 0000	20H
JMP	2085H	1100 0011	C3H
		1000 0101	85H
		0010 0000	20H

RECOGNISING THE LENGTH OF AN INSTRUCTION

1. One-byte instruction – A mnemonics followed by a letter (or two letters) representing a register (such as A,B,C,D,H,L,M, and SP) is a one byte instruction. Instructions in which register are implicit are also one byte instruction.

   Examples: (i) MOV A, B (ii) DCX H (iii) CMA

2. Two-byte instruction – A mnemonics followed by 8-bits (byte) is a two byte instruction.

   Examples: (i) MVI A, 8-bit (ii) ADI 8-bit

3. Three-byte instructions – A mnemonic followed by 16-bit (word) is a three byte instruction.

Example: (i) LXI B, 16-bit (data) (ii) LDA 16-bit (address)

OPCODE FORMAT

In the design of the 8085 MP chip, all operations, registers and status flags are identified with a specific code. The lists are given below.

CODE	REGISTERS	CODE	REGISTER PAIRS
000	B	00	BC
001	C	01	DE
010	D	10	HL
011	E	11	AF OR SP
100	H		
101	L		
111	A		
110	Reserved for memory related operation		

Some of the examples are given below showing how the op-code is calculated.

1. **MOV C,A**

Move (copy) the content		: 01
To register C		: 001
From register A	: 111	
Combined binary instruction		: 01001111= 4FH

2. **ADD B**

Add	: 10000
Register B	: 000
To A	: Implicit
Combined binary instruction	: 10000000 = 80H

INSTRUCTION SETS

An Instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described here are of Intel 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufactures. The programmer can write a program in assembly language using these instructions.

These instructions have been classified into the following groups:

1. Data Transfer Group
2. Arithmetic Group
3. Logical Group
4. Branch Control Group
5. I/O and Machine Control Group

Data Transfer group

Instructions, which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA etc.

These operations simply **COPY** the data from the source to the destination.

- They transfer: Data between registers.
- Data Byte to a register or memory location.
- Data between a memory location and a register.
- Data between an I\O Device and the accumulator.
- The data in the source is not changed.

Opcode	Operand	Explanation of Instruction	Description
MOV	**Rd, Rs** **M, Rs** **Rd, M**	Copy from source(Rs) to destination(Rd)	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. **Example: MOV B, C or MOV B, M**
MVI	**Rd, data** **M, data**	Move immediate 8-bit	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the

			contents of the HL registers. **Example: MVI B, 57H or MVI M, 57H**
LDA	**16-bit address**	Load accumulator	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. **Example: LDA 2034H**
LDAX	**B/D Reg. pair**	Load accumulator indirect	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. **Example: LDAX B**
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand. E**xample: LXI H, 2034H or LXI H, XYZ**
LHLD	**16-bit address**	Load H and L registers direct	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. **Example: LHLD 2040H**
STA	**16-bit address**	16-bit address	The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. **Example: STA 4350H**
STAX	**Reg. pair**	Store accumulator indirect	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

			Example: STAX B
SHLD	**16-bit address**	Store H and L registers direct	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address. **Example: SHLD 2470H**
XCHG	**none**	Exchange H and L with D and E	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. **Example: XCHG**
SPHL	**none**	Copy H and L registers to the stack pointer	The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered. **Example: SPHL**
XTHL	**none**	Exchange H and L with top of stack	The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered. **Example: XTHL**
PUSH	**Reg. pair**	Push register pair onto stack	The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the highorder register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E,

			L, flags) are copied to that location. **Example: PUSH B or PUSH A**
POP	**Reg. pair**	Pop off stack to register pair	The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1. **Example: POP H or POP A**
OUT	**8-bit port address**	Output data from accumulator to a port with 8-bit address	The contents of the accumulator are copied into the I/O port specified by the operand. **Example: OUT F8H**
IN	**8-bit port address**	Input data to accumulator from a port with 8-bit address	The contents of the input port designated in the operand are read and loaded into the accumulator. **Example: IN 8CH**

Exchange the contents of memory locations 2000H and 4000H

Program 1:

```
LDA 2000H     : Get the contents of memory location 2000H into accumulator
MOV B, A      : Save the contents into B register
LDA 4000H     : Get the contents of memory location 4000Hinto accumulator
STA 2000H     : Store the contents of accumulator at address 2000H
MOV A, B      : Get the saved contents back into A register
STA 4000H     : Store the contents of accumulator at address 4000H
```

Program 2:

```
LXI H 2000H     : Initialize HL register pair as a pointer to memory location 2000H.
LXI D 4000H     : Initialize DE register pair as a pointer to memory location 4000H.
MOV B, M        : Get the contents of memory location 2000H into B register.
LDAX D          : Get the contents of memory location 4000H into A register.
MOV M, A        : Store the contents of A register into memory location 2000H.
MOV A, B        : Copy the contents of B register into accumulator.
STAX D          : Store the contents of A register into memory location 4000H.
HLT             : Terminate program execution.
```

Arithmetic group

The instructions of this group perform arithmetic operations such as addition, subtraction; increment or decrement of the content of a register or memory. Examples are: ADD, SUB, INR, DAD etc.

Opcode	Operand	Explanation of Instruction	Description
ADD	**R** **M**	Add register or memory, to accumulator	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. **Example: ADD B or ADD M**
ADC	**R** **M**	Add register to accumulator with carry	The contents of the operand (register or memory) and M the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. **Example: ADC B or ADC M**
ADI	**8-bit data**	Add immediate to accumulator	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

			Example: ADI 45H
ACI	**8-bit data**	Add immediate to accumulator with carry	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. **Example: ACI 45H**
LXI	**Reg. pair, 16-bit data**	Load register pair immediate	The instruction loads 16-bit data in the register pair designated in the operand.E**xample: LXI H, 2034H or LXI H, XYZ**
DAD	**Reg. pair**	Add register pair to H and L registers	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. **Example: DAD H**
SUB	**R** **M**	Subtract register or memory from accumulator	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example: SUB B or SUB M**
SBB	**R** **M**	Subtract source and borrow from accumulator	The contents of the operand (register or memory) and M the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction. **Example: SBB B or SBB M**
SUI	**8-bit data**	Subtract immediate from accumulator	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction. **Example: SUI 45H**
SBI	**8-bit data**	Subtract immediate from accumulator with borrow	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E. **Example: XCHG**

INR	**R** **M**	Increment register or memory by 1	The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example: INR B or INR M**
INX	**R**	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and the result is stored in the same place. **Example: INX H**
DCR	**R** **M**	Decrement register or memory by 1	The contents of the designated register or memory are M decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers. **Example: DCR B or DCR M**
DCX	**R**	Decrement register pair by 1	The contents of the designated register pair are decremented by 1 and the result is stored in the same place. **Example: DCX H**
DAA	**none**	Decimal adjust accumulator	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits. If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.**Example: DAA**

Statement: Add the 16-bit number in memory locations 4000H and 4001H to the 16-bit number in memory locations 4002H and 4003H. The most significant eight bits of the two numbers to be added are in memory locations 4001H and 4003H. Store the result in

memory locations 4004H and 4005H with the most significant byte in memory location 4005H.

Using ADD and ADC instruction

```
LHLD 4000H      : Get first I6-bit number in HL
XCHG            : Save first I6-bit number in DE
LHLD 4002H      : Get second I6-bit number in HL
MOV A, E        : Get lower byte of the first number
ADD L           : Add lower byte of the second number
MOV L, A        : Store result in L register
MOV A, D        : Get higher byte of the first number
ADC H           : Add higher byte of the second number with CARRY
MOV H, A        : Store result in H register
SHLD 4004H      : Store I6-bit result in memory locations 4004H and 4005H.
HLT             : Terminate program execution
```

Using DAD instruction

```
LHLD 4000H      : Get first I6-bit number
XCHG            : Save first I6-bit number in DE
LHLD 4002H      : Get second I6-bit number in HL
DAD D           : Add DE and HL
SHLD 4004H      : Store I6-bit result in memory locations 4004H and 4005H.
HLT             : Terminate program execution
```

Logical group

The Instructions under this group perform logical operation such as AND, OR, compare, rotate etc. Examples are: ANA, XRA, ORA, CMP, and RAL etc.

Opcode	Operand	Explanation of Instruction	Description

CMP	**R**
	M
		Compare register or memory with accumulator	The contents of the operand (register or memory) are M compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < (reg/mem): carry flag is set
			if (A) = (reg/mem): zero flag is set

			if (A) > (reg/mem): carry and zero flags are reset **Example: CMP B or CMP M**

CPI	**8-bit data**	Compare immediate with accumulator	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
			
			if (A) < data: carry flag is set
			if (A) = data: zero flag is set
			if (A) > data: carry and zero flags are reset
			
			Example: CPI 89H

ANA	**R** **M**	Logical AND register or memory with accumulator	The contents of the accumulator are logically ANDed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. **Example: ANA B or ANA M**
ANI	**8-bit data**	Logical AND immediate with accumulator	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set. **Example: ANI 86H**
XRA	**R** **M**	Exclusive OR register or memory with accumulator	The contents of the accumulator are Exclusive ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: XRA B or XRA M**
XRI	**8-bit**	Exclusive OR	The contents of the accumulator are Exclusive ORed with the 8-bit

	data	immediate with accumulator	data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: XRI 86H**
ORA	**R** **M**	Logical OR register or memory with accumulator	The contents of the accumulator are logically ORed with M the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: ORA B or ORA M**
ORI	**8-bit data**	Logical OR immediate with accumulator	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset. **Example: ORI 86H**
RLC	**none**	Rotate accumulator left	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected. **Example: RLC**
RRC	**none**	Rotate accumulator right	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected. **Example: RRC**
RAL	**none**	Rotate accumulator left through carry	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected. **Example: RAL**

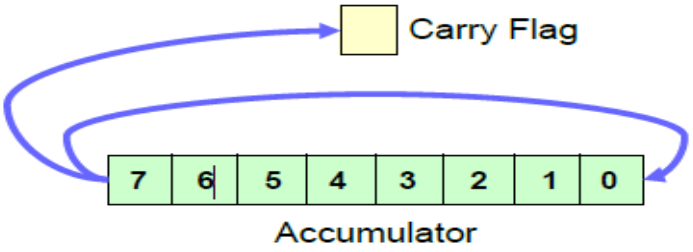
RAR	**none**	Rotate accumulator right through carry	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected. **Example: RAR**
CMA	**none**	Complement accumulator	The contents of the accumulator are complemented. No flags are affected. **Example: CMA**
CMC	**none**	Complement carry	The Carry flag is complemented. No other flags are affected. **Example: CMC**
STC	**none**	Set Carry	Set Carry **Example: STC**

Rotate: Rotate the contents of the accumulator one position to the left or right.

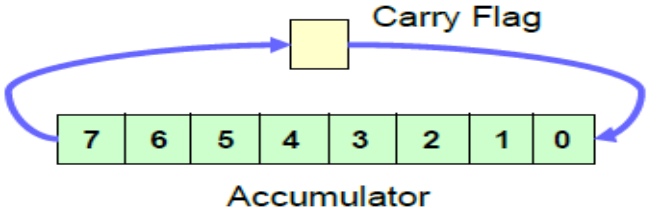
- **RLC**: Rotate the accumulator left. Bit 7 goes to bit 0 and the Carry flag.
- **RAL**: Rotate the accumulator left through the carry. Bit 7 goes to the carry and carry goes to bit 0.
- **RRC**: Rotate the accumulator right. Bit 0 goes to bit 7 and the Carry flag.
- **RAR**: Rotate the accumulator right through the carry. Bit 0 goes to the carry and carry goes to bit 7.

RLC vs. RLA

- RLC



- RAL



Program: Find the 2's complement of the number stored at memory location 4200H and store the complemented number at memory location 4300H.

```
LDA 4200H      : Get the number
CMA            : Complement the number
ADI, 01 H      : Add one in the number
STA 4300H      : Store the result
HLT            : Terminate program execution
```

Branch Control group

This group includes the instructions for conditional and unconditional jump, subroutine call and return, and restart. Examples are: JMP, JC, JZ, CALL, CZ, RST etc.

Opcode	Operand	Explanation of Instruction	Description
JMP	**16-bit address**	Jump unconditionally	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. **Example: JMP 2034H or JMP XYZ**
Opcode: **JC** / Description: **Jump on Carry** / Flag Status: **CY = 1**	**16-bit address**	Jump conditionally	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag

JNC	**Jump on no Carry**	**CY = 0**			of the PSW as described below. **Example: JZ 2034H or JZ XYZ**
JP	**Jump on positive**	**S = 0**			
JM	**Jump on minus**	**S = 1**			
JZ	**Jump on zero**	**Z = 1**			
JNZ	**Jump on no zero**	**Z = 0**			
JPE	**Jump on parity even**	**P = 1**			
JPO	**Jump on parity odd**	**P = 0**			
Opcode	Description	Flag Status	**16-bit address**	Unconditional subroutine call	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack. **Example: CALL 2034H or CALL XYZ**
CC	**Call on Carry**	**CY = 1**			
CNC	**Call on no Carry**	**CY = 0**			
CP	**Call on positive**	**S = 0**			
CM	**Call on minus**	**S = 1**			
CZ	**Call on zero**	**Z = 1**			
CNZ	**Call on no zero**	**Z = 0**			
CPE	**Call on parity even**	**P = 1**			
CPO	**Call on parity odd**	**P = 0**			
RET			**none**	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter,and program execution begins at the new address. **Example: RET**
Opcode	Description	Flag Status	**none**	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are
RC	**Return on Carry**	**CY = 1**			
RNC	**Return on no Carry**	**CY = 0**			

<table>
<tr>
<td>
<table>
<tr><td>RP</td><td>Return on positive</td><td>S = 0</td></tr>
<tr><td>RM</td><td>Return on minus</td><td>S = 1</td></tr>
<tr><td>RZ</td><td>Return on zero</td><td>Z = 1</td></tr>
<tr><td>RNZ</td><td>Return on no zero</td><td>Z = 0</td></tr>
<tr><td>RPE</td><td>Return on parity even</td><td>P = 1</td></tr>
<tr><td>RPO</td><td>Return on parity odd</td><td>P = 0</td></tr>
</table>
</td>
<td></td>
<td></td>
<td>copied into the program counter, and program execution begins at the new address.

Example: RZ</td>
</tr>
<tr>
<td>PCHL</td>
<td>none</td>
<td>Load program counter with HL contents</td>
<td>The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.

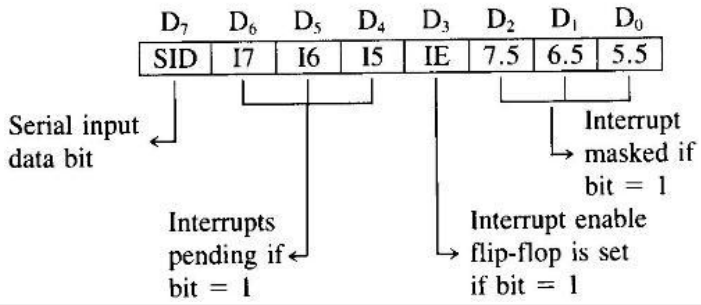
Example: PCHL</td>
</tr>
<tr>
<td>RST</td>
<td>0-7</td>
<td>Restart</td>
<td>The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:
<table>
<tr><td>Instruction</td><td>Restart Address</td></tr>
<tr><td>RST 0</td><td>0000H</td></tr>
<tr><td>RST1</td><td>0008H</td></tr>
<tr><td>RST 2</td><td>0010H</td></tr>
<tr><td>RST 3</td><td>0018H</td></tr>
<tr><td>RST 4</td><td>0020H</td></tr>
<tr><td>RST 5</td><td>0028H</td></tr>
<tr><td>RST 6</td><td>0030H</td></tr>
</table>
</td>
</tr>
</table>

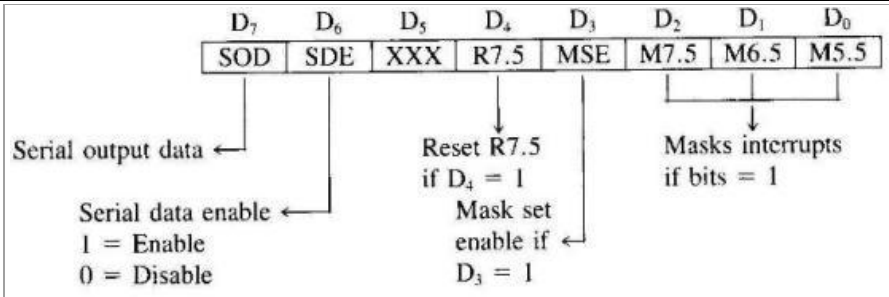
			RST 7 **0038H** The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are: Interrupt / Restart Address **TRAP** **0024H** **RST 5.5** **002CH** **RST 6.5** **0034H** **RST 7.5** **003CH**

I/O and Machine control group

This group includes the instructions for input/output ports, stack and machine control. Examples are: IN, OUT, PUSH, POP, and HLT etc.

Opcode	Operand	Explanation of Instruction	Description
NOP	**none**	No operation	No operation is performed. The instruction is fetched and decoded. However no operation is executed. **Example: NOP**
HLT	**none**	Halt and enter wait state	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state. **Example: HLT**
DI	**none**	Disable interrupts	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.

			Example: DI
EI	**none**	Enable interrupts	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flipflop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP). **Example: EI**
RIM	**none**	Read interrupt mas	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations. **Example: RIM** 
SIM	**none**	Set interrupt mask	This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows. **Example: SIM**



- SOD — Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit $D_6 = 1$.
- SDE — Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- XXX — Don't Care
- R7.5 — Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- MSE — Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- M7.5 — D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- M6.5 — D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- M5.5 — D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

Addressing modes of 8085

- To perform any operation, we have to give the corresponding instructions to the microprocessor.
- In each instruction, programmer has to specify 3 things:
  - Operation to be performed.
  - Address of source of data.
  - Address of destination of result.
- The method by which the address of source of data or the address of destination of result is given in the instruction is called **Addressing modes.**
- The term addressing mode refers to the way in which the operand of the instruction is specified.

Types of Addressing Modes

Intel 8085 uses the following addressing modes:

1. Direct Addressing Mode
2. Register Addressing Mode
3. Register Indirect Addressing Mode
4. Immediate Addressing Mode
5. Implicit Addressing Mode

- **Direct Addressing Mode**

  In this mode, the address of the operand is given in the instruction itself. For example

  - LDA 3456H
  - STA 6785H
  - LHLD 4532H

- **Register Addressing Mode**

  In this mode, the operand is in general purpose register. For example

  - MOV A,B
  - ANA B
  - SBB C

- **Register Indirect Addressing Mode**

  In this mode, the address of operand is specified by a register pair. For example

  - MOV A,M
  - LDAX B
  - STAX D

  .

- **Immediate Addressing Mode**

  In this mode, the operand is specified within the instruction itself. For example

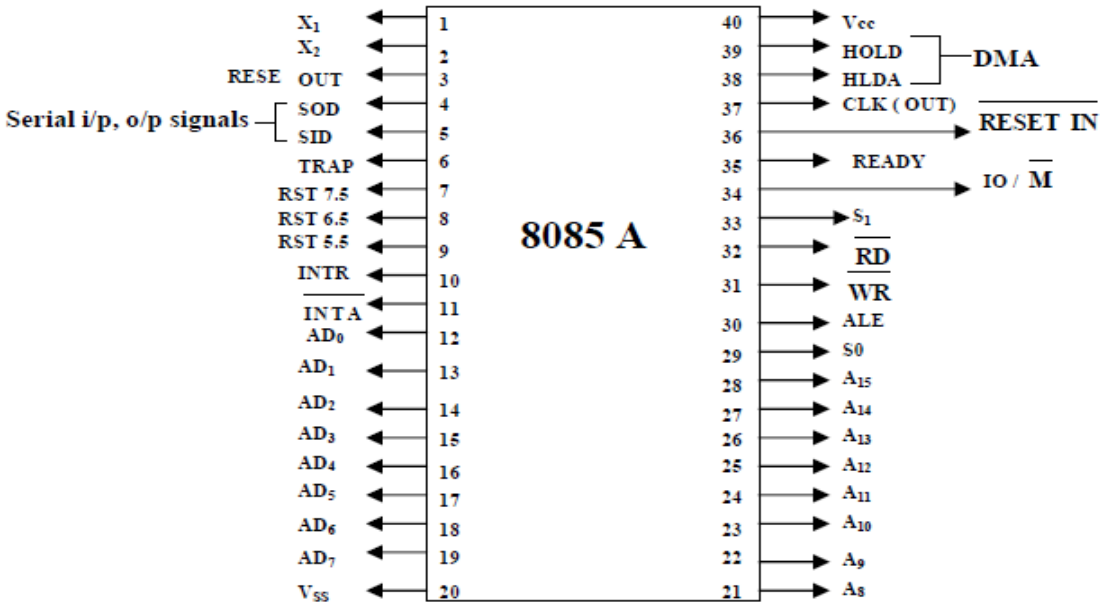
  - MVI A,65H
  - ADI 45H
  - SUI 42H

- **Implicit Addressing Mode**

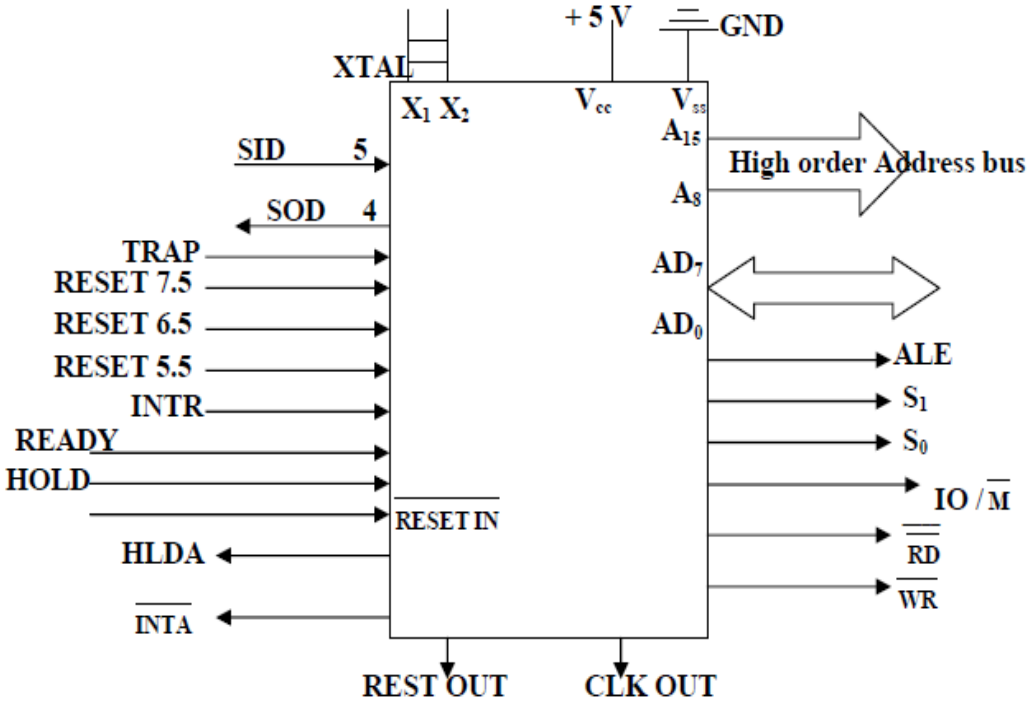
  If address of source of data as well as address of destination of result is fixed, then there is no need to give any operand along with the instruction. For example

  - CMA
  - DAA

Pin Diagram of 8085



(Pin diagram of 8085)



(Signal groups of 8085)

The total 40 pins of microprocessor is divided by following group.

1. Power supply and frequency signal
2. Address and data bus
3. Control and Status signals
4. Externally and Peripheral Initiated signals
5. Serial Communication signals

Power supply and frequency signals

Vcc and Vss (Pin 40 and 20):

- Vcc (pin. 40) is +5v pin and Vss (pin.20) is ground pin.

X1, X2 (Pin 1 and 3, Input)

- Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

Address and Data bus

Address Bus (Pin 21 to 28, output):

- The pins A8-A15 denote the address bus. They are used for the most significant bit of memory address.
- These pins carry the higher order of address bus.
- The address is sent from microprocessor to memory.
- These 8 pins are switched to high impedance state during HOLD and RESET mode.

Address/Data Bus (Pin 12 to 19, input/output):

- AD0-AD7 constitutes the Address/Data bus. They are time multiplexed. These pins are used for least significant bits of address bus in the first machine clock cycle and used as data bus for second and third clock cycle.
- These pins serve the dual purpose of transmitting lower order address and data byte.
- During 1st clock cycle, these pins act as lower half of address. In remaining clock cycles, these pins act as data bus.
- The separation of lower order address and data is done by address latch.

Control and Status signals

S0 (Pin 29, output) and S1 (Pin 33,output)

- They tell the current operation which is in progress in 8085.

S_0	S_1	Operation
0	0	Halt
0	1	Write
1	0	Read
1	1	Opcode Fetch

IO/M*(Pin 34,output)

- This pin tells whether I/O or memory operation is being performed.
  - If IO/M = 1 then, I/O operation is being performed.
  - If IO/M = 0 then, Memory operation is being performed.

Table below showing IO/M, S0, S1 and corresponding operations.

Operations	$IO/\overline{M}$	S_0	S_1
Opcode Fetch	0	1	1
Memory Read	0	1	0
Memory Write	0	0	1
I/O Read	1	1	0
I/O Write	1	0	1
Interrupt Ack.	1	1	1
Halt	High Impedance	0	0

RD* (Pin 34, output)

- RD stands for Read.
- It is an active low signal. It is a control signal used for Read operation either from memory or from Input device.
- A low signal indicates that data on the data bus must be placed either from selected memory location or from input device.

WR* (Pin 31, output)

- WR stands for Write.
- It is also active low signal.
- It is a control signal used for Write operation either into memory or into output device.
- A low signal indicates that data on the data bus must be written into selected

memory location or into output device.

ALE (Pin 30, output)

- It is used to enable Address Latch.
- It indicates whether bus functions as address bus or data bus.
- If ALE = 1 then, bus functions as address bus.
- If ALE = 0 then, bus functions as data bus.

Externally and Peripheral initiated signals

Interrupts: Interrupt is a process by which microprocessor stops executing the main program and do some other brief program stored in pre-defined memory location and after finishing that it will again come back to the main program.

- It means *interrupting* the normal execution of the microprocessor.
- When microprocessor receives interrupt signal, it discontinues whatever it was executing.
- It starts executing new program indicated by the interrupt signal.
- Interrupt signals are generated by external peripheral devices.
- After execution of the new program, microprocessor goes back to the previous program.
- The processor has 5 interrupts. They are presented below in the order of their priority (from highest to lowest): TRAP, RST 7.5, RST 6.5, RST 5.5, INTR

TRAP (Pin 6, input):

- It is an non-maskable interrupt.
- It has the highest priority.
- It cannot be disabled.
- It is both edge and level triggered.
- It means TRAP signal must go from low to high and must remain high for a certain period of time.
- TRAP is usually used for power failure and emergency shutoff.

RST 7.5 (Pin 7, input)

- It is a maskable interrupt.
- It has the second highest priority.
- It is positive edge triggered only.
- The internal flip-flop is triggered by the rising edge.
- The flip-flop remains high until it is cleared by RESET IN.

RST 6.5(Pin 8, Input)

- It is a maskable interrupt.
- It has the third highest priority.
- It is level triggered only.
- The pin has to be held high for a specific period of time.

- RST 6.5 can be enabled by EI instruction.
- It can be disabled by DI instruction.

RST 5.5 (pin 9, Input)

- It is a maskable interrupt.
- It has the fourth highest priority.
- It is also level triggered.
- The pin has to be held high for a specific period of time.
- This interrupt is very similar to RST 6.5.

INTR (pin 10, input)

- It is a maskable interrupt.
- It has the lowest priority.
- It is also level triggered.
- It is a general purpose interrupt.
- By general purpose we mean that it can be used to vector microprocessor to any specific subroutine having any address.

INTA* (pin 11, Output)

- It stands for interrupt acknowledge.
- It is an outgoing signal.
- It is an active low signal.
- Low output on this pin indicates that microprocessor has acknowledged the INTR request.

READY (pin 35, Input)

- This pin is used to synchronize slower peripheral devices with fast microprocessor.
- A low value causes the microprocessor to enter into ***wait state***.
- The microprocessor remains in wait state until the input at this pin goes high.

HOLD (Pin 38, Input)

- HOLD pin is used to request the microprocessor for DMA transfer.
- A high signal on this pin is a request to microprocessor to relinquish the hold on buses.
- This request is sent by DMA controller.
- Intel 8257 and Intel 8237 are two DMA controllers.

HLDA (Pin 39, Output)

- HLDA stands for Hold Acknowledge.
- The microprocessor uses this pin to acknowledge the receipt of HOLD signal.
- When HLDA signal goes high, address bus, data bus, RD, WR, IO/M pins are ***tri-stated***.
- This means they are cut-off from external environment.
- The control of these buses goes to DMA Controller.

- Control remains at DMA Controller until HOLD is held high.
- When HOLD goes low, HLDA also goes low and the microprocessor takes control of the buses.

RESET IN* (Pin 36, Input)

- It is used to reset the microprocessor.
- It is active low signal. ◦ When the signal on this pin is low for at least 3 clocking cycles, it forces the microprocessor to reset itself.
- Resetting the microprocessor means:
    - Clearing the PC and IR.
    - Disabling all interrupts (except TRAP).
    - Disabling the SOD pin.
    - All the buses (data, address, control) are ***tristated***.
    - Gives HIGH output to RESET OUT pin.

RESET OUT (Pin 3, output)

- It is used to reset the peripheral devices and other ICs on the circuit.
- It is an output signal.
- It is an active high signal.
- The output on this pin goes high whenever RESET IN is given low signal.
- The output remains high as long as RESET IN is kept low.

Serial I/O signals

SID (Pin 4, Input)

- Serial Input Data
- It takes 1 bit input from serial port of 8085.
- Stores the bit at the 8^{th} position (MSB) of the Accumulator.
- RIM (Read Interrupt Mask) instruction is used to transfer the bit.

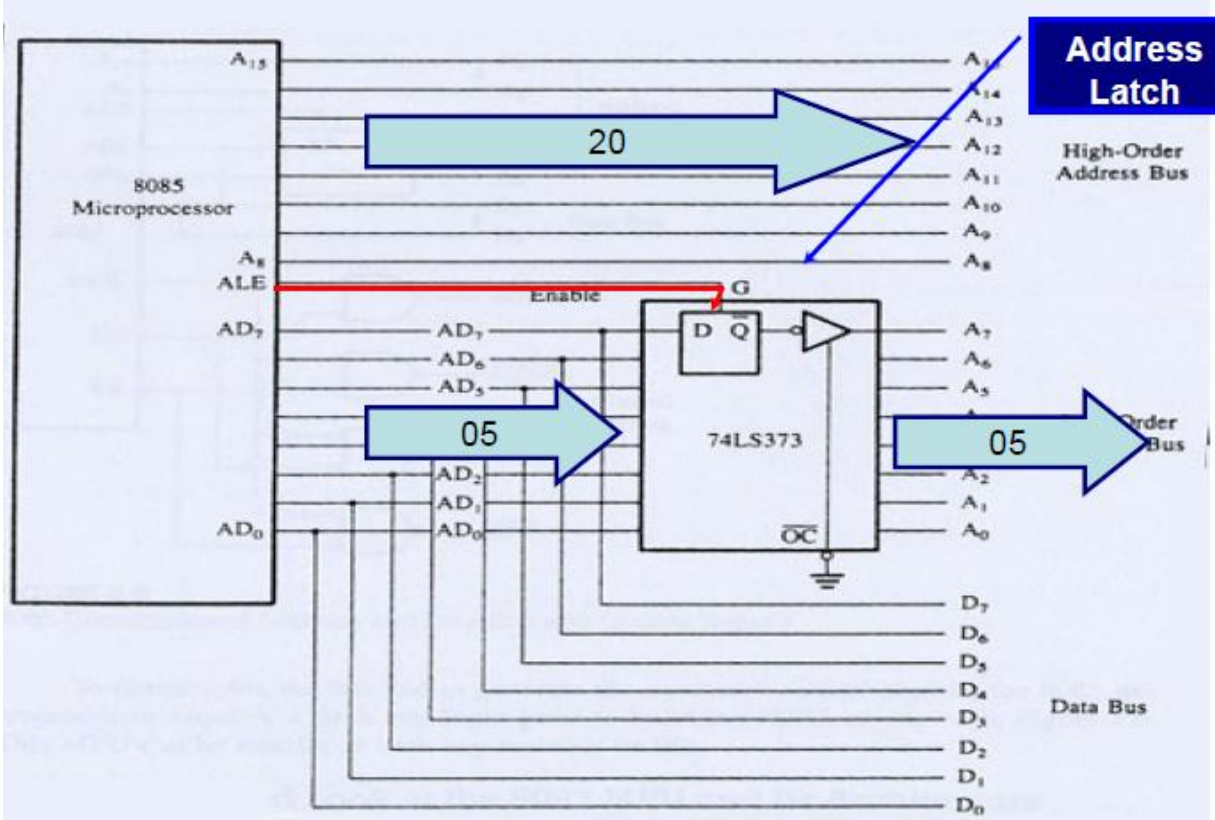
SOD (Pin 5, Output)

- Serial Output Data
- It takes 1 bit from Accumulator to serial port of 8085.
- Takes the bit from the 8^{th} position (MSB) of the Accumulator.
- SIM (Set Interrupt Mask) instruction is used to transfer the bit.

Demultiplexing of Busses

It is very important to reduce number of pins in an IC. As we know address in 8085 contains 16 bits, and data contains 8 pins, we need 24 pins in IC to be used as address and data pins, rather than this designer has a different approach to reduce the usage of pins for address & data. Only 16 pins are used totally for address and data, in which 8 pins (AD_0-AD_7) are combined, used to generate address & data. Microprocessor generates both data & addresses one same 8 pins. The thing is to resolve both address

& data from these pins. This process is achieved through demultiplexing the address & data signals



Procedure: - Every Instruction has some machine cycles to complete its executions sometime called T states. Whenever an instruction is executed by MPU first of all MPU sends **ALE** signal to address latch IC to enable all D Latches to receive new address from MPU now (in first T state) Microprocessor generates the address on Address Bus, half portion of address (lower order address) is generated on AD_0-AD_7 . This Address bits are captured by D latches and stored in. Now during next cycles say T2 T3 and so on, MPU can use AD_0-AD_7 as Data Bus to send receives data. During this period the initially generated Address is also available at output pins of D Latches .The IC 74LS31 is used as Address Latch it contains 8 D Latches to store lower half of address.(8 bits).

Memory Interfacing

Generally in all cases, total 64 KB of memory may not be interfaced with microprocessor always. The size of the memory and no. of memory is to be interfaced with the microprocessor depend the application and availability.

Memory IC EPROM/RAM	Capacity	Number of address pins	Number of data pins
2708/6208	1kb	10	8
2716/6216	2kb	11	8
2732/6232	4kb	12	8
2764/6264	8kb	13	8
27256/62256	32kb	15	8
27512/62512	64kb	16	8
27010/62128	128kb	17	8
27020/62138	256kb	18	8
27040/62148	512kb	19	8

Table - Number of Address Pins and Data Pins in Memory ICs

Procedure

a. The no. of address lines required to identify each location of memory is directly connected to the same amount of address lines of microprocessor.

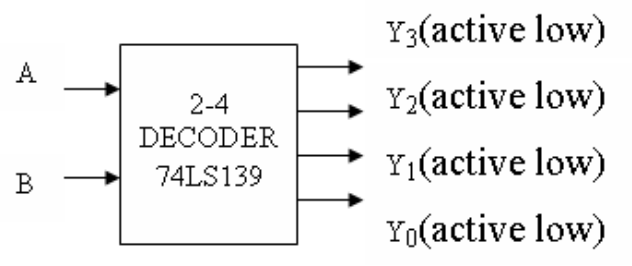
b. The remaining address lines of microprocessor are used to decode for the chip selection lines of memory.

If more than one memory is to be interfaced with microprocessor then a decoder is used to activate the chip selection line of memory.

Decoder:

It is used to select the memory chip of processor during the execution of a program. No of IC's used for decoder is,

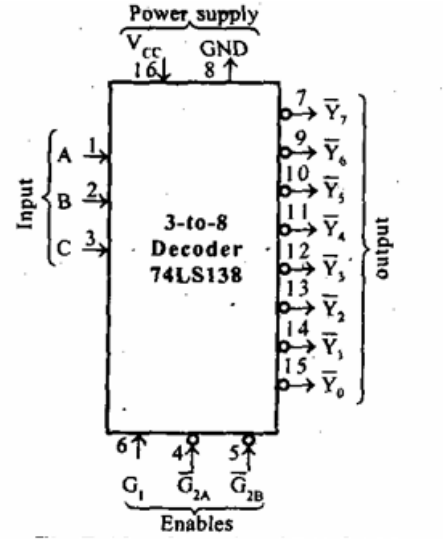
- **2-4 decoder (74LS139)**



Input		Output			
B	A	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	1	1	1	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1

Fig - Block diagram and Truth table of 2-4 decoder

- **3-8 decoder (74LS138)**



Input			Output							
C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	0	0	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	0	1
0	1	0	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	1	1	0	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1
1	1	0	1	0	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1

Fig - Block diagram and Truth table of 3-8 decoder

Example-1

Consider a system in which the full memory space 64kb is utilized for EPROM memory. Interface the EPROM with 8085 processor.

- The memory capacity is 64 Kbytes. i.e
- 2^n = 64 x 1000 bytes where n = address lines.
- So, n = 16.
- In this system the entire 16 address lines of the processor are connected to address input pins of memory IC in order to address the internal locations of memory.
- The chip select (CS) pin of EPROM is permanently tied to logic low (i.e., tied to ground).
- Since the processor is connected to EPROM, the active low RD pin is connected to active low output enable pin of EPROM.
- The range of address for EPROM is 0000H to FFFFH.

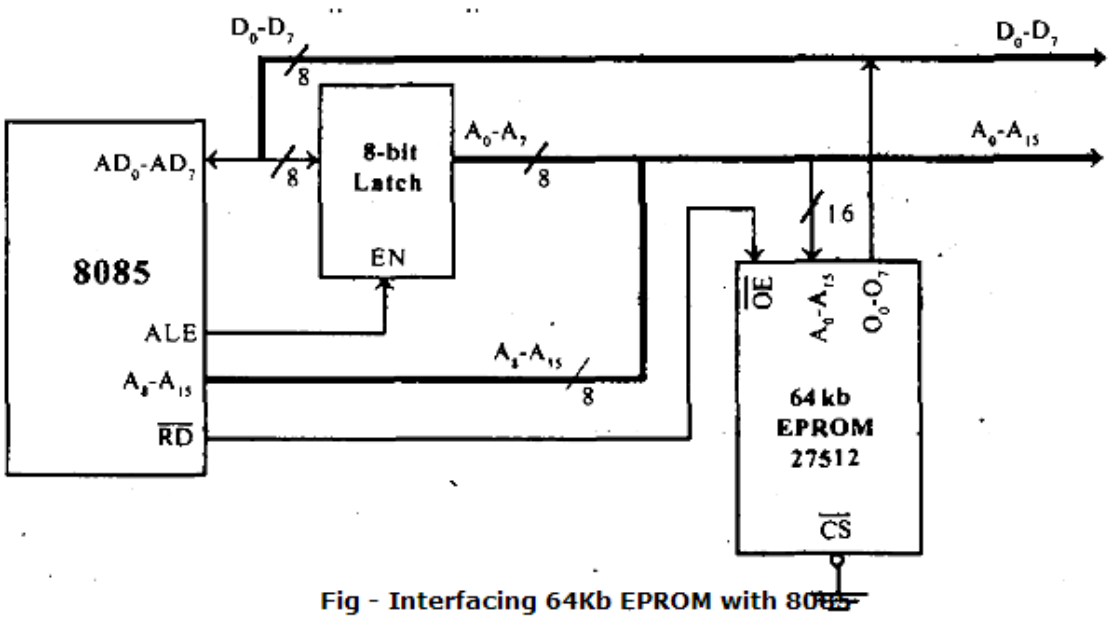


Fig - Interfacing 64Kb EPROM with 8085

Example-2

Consider a system in which the available 64kb memory space is equally divided between EPROM and RAM. Interface the EPROM and RAM with 8085 processor.

- Implement 32kb memory capacity of EPROM using single IC 27256.
- 32kb RAM capacity is implemented using single IC 62256.
- The 32kb memory requires 15 address lines and so the address lines A0 - A14 of the processor are connected to 15 address pins of both EPROM and RAM.
- The unused address line A15 is used as to chip select. If A15 is 1, it select RAM and If A15 is 0, it select EPROM.
- Inverter is used for selecting the memory.
- The memory used is both Ram and EPROM, so the low RD and WR pins of processor are connected to low WE and OE pins of memory respectively.
- The address range of EPROM will be 0000H to 7FFFH and that of RAM will be 7FFFH to FFFFH.

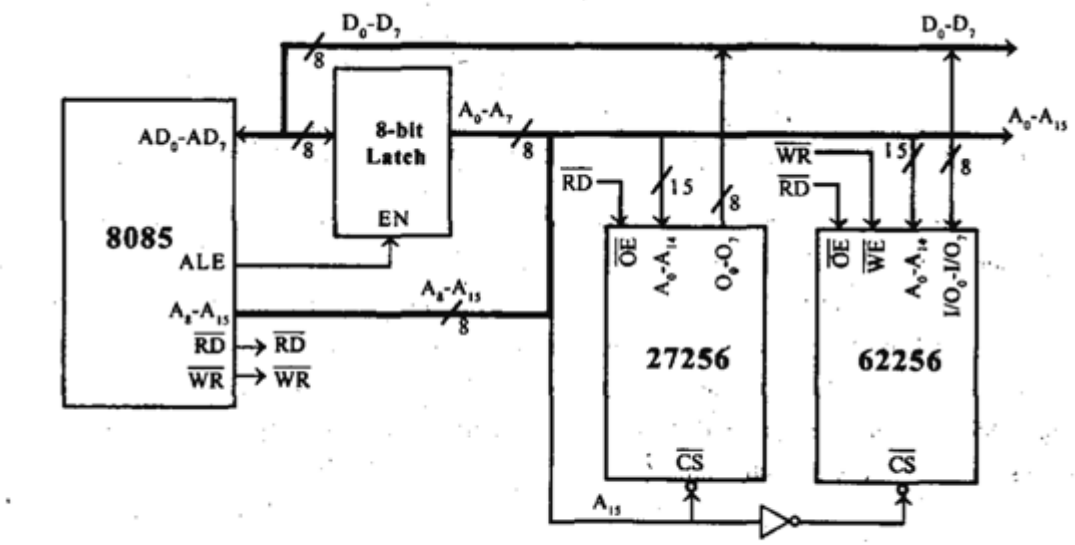


Fig - Interfacing 32Kb EPROM and 32Kb RAM with 8085

Example-3

Consider a system in which 32kb memory space is implemented using four numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 32Kb. So, let two number of 8kb n memory be EPROM and the remaining two numbers be RAM.
- Each 8kb memory requires 13 address lines and so the address lines A0- A12 of the processor are connected to 13 address pins of all the memory.
- The address lines and A13 - A14 can be decoded using a 2-to-4 decoder to generate four chip select signals.
- These four chip select signals can be used to select one of the four memory IC at any one time.
- The address line A15 is used as enable for decoder.
- The simplified schematic memory organization is shown.

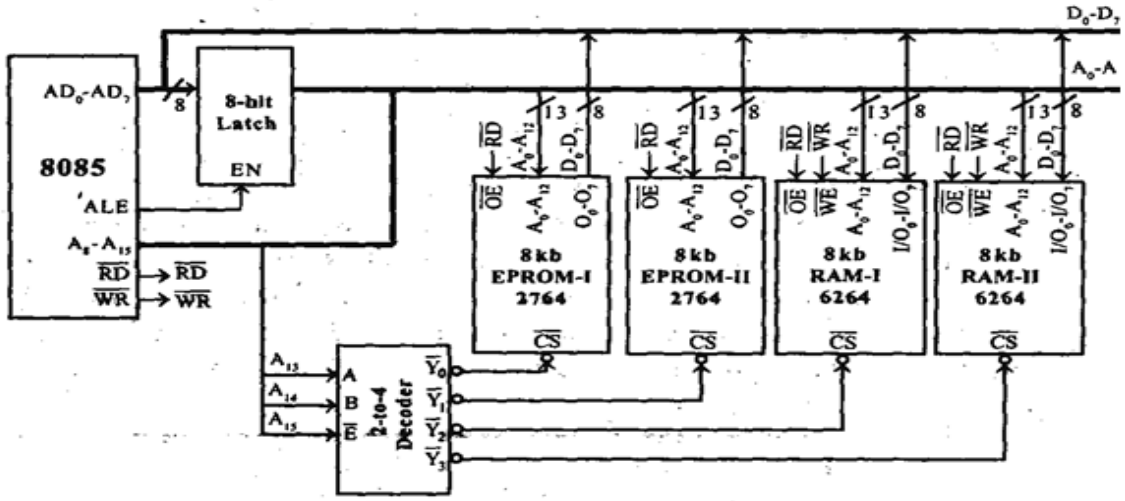


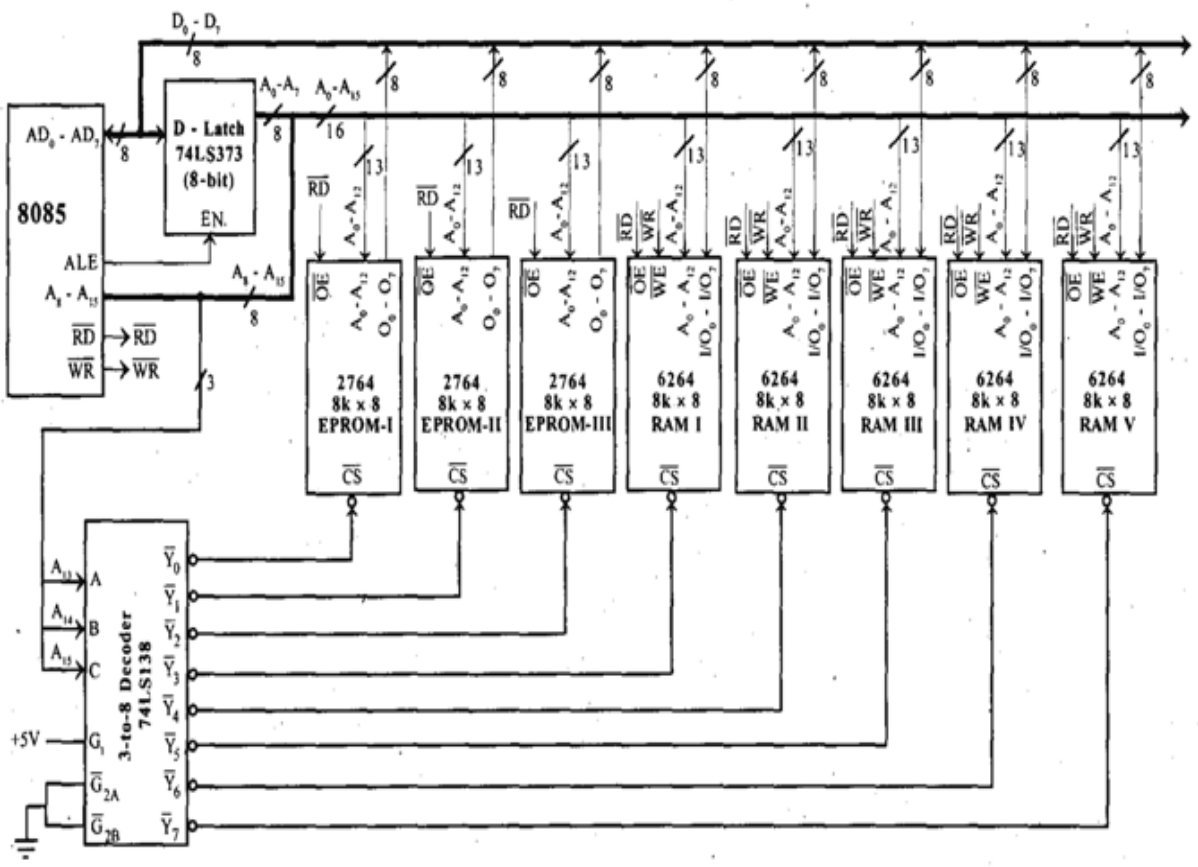
Fig - Interfacing 16Kb EPROM and 16Kb RAM with 8085

Device	Binary address																Hexa address
	Decoder enable/input			Input to address pins of memory IC													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
8kb EPROM - I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
8kb EPROM - II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
8kb RAM - I	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
8kb RAM -II	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF

Example-4

Consider a system in which the 64kb memory space is implemented using eight numbers of 8kb memory. Interface the EPROM and RAM with 8085 processor.

- The total memory capacity is 64Kb. So, let 4 numbers of 8Kb EPROM and 4 numbers of 8Kb RAM.
- Each 8kb memory requires 13 address lines. So the address line A0 - A12 of the processor are connected to 13address pins of all the memory lCs.
- The address lines A13, A14 and A]5 are decoded using a 3-to-8 coder to generate eight chip select signals. These eight chip select signals can be used to select one of the eight memories at any one time.
- The memory interfacing is shown in following figure.



Memory IC chip	Binary address																Hexa address
	Decoder input			Input to memory address pins													
	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
EPROM I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1FFF
EPROM II	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2000
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	2001
	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	2002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3FFF
EPROM III	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4000
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4001
	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	4002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	5FFF
RAM I	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6000
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	6001
	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	6002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7FFF
RAM II	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	8001
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	8002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	9FFF
RAM III	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	A000
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	A001
	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	A002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	BFFF
RAM IV	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C000
	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	C001
	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	C002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	DFFF
RAM V	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	E000
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	E001
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	E002
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF

INTERRUPT

- Interrupt is signals send by an external device to the processor, to request the processor to perform a particular task or work.
- Mainly in the microprocessor based system the interrupts are used for data transfer between the peripheral and the microprocessor.
- The processor will check the interrupts always at the 2nd T-state of last machine cycle.
- If there is any interrupt it accept the interrupt and send the INTA (active low) signal to the peripheral.
- The vectored address of particular interrupt is stored in program counter.
- The processor executes an interrupt service routine (ISR) addressed in program counter.
- It returned to main program by RET instruction.

Types of Interrupts:

It supports two types of interrupts.

- Software
- Hardware

Software interrupts:

- The software interrupts are program instructions. These instructions are inserted at desired locations in a program.
- The 8085 has eight software interrupts from RST 0 to RST 7. The vector address for these interrupts can be calculated as follows.

- Interrupt number * 8 = vector address

For RST 5.5 * 8 = 40 = 28H
Vector address for interrupt RST 5 is 0028H

The Table shows the vector addresses of all interrupts.

Interrupt	Vector address
RST 0 RST 1	0000_H 0008_H
RST 2 RST 3	0010_H 0018_H
RST 4 RST 5	0020_H 0028_H
RST 6 RST 7	0030_H 0038_H

Hardware interrupts:

- An external device initiates the hardware interrupts and placing an appropriate signal at the interrupt pin of the processor.
- If the interrupt is accepted then the processor executes an interrupt service routine.

The 8085 has five hardware interrupts

(1) TRAP (2) RST 7.5 (3) RST 6.5 (4) RST 5.5 (5) INTR

Interrupt	Vector address
RST 7.5	$003C_H$
RST 6.5	0034_H
RST 5.5	$002C_H$
TRAP	0024_H

TRAP:

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- TRAP bas the highest priority and vectored interrupt.

- TRAP interrupt is edge and level triggered. This means that the TRAP must go high and remain high until it is acknowledged.

- In sudden power failure, it executes a ISR and send the data from main memory to backup memory.

- The signal, which overrides the TRAP, is HOLD signal. (i.e., If the processor receives HOLD and TRAP at the same time then HOLD is recognized first and then TRAP is recognized).

- There are two ways to clear TRAP interrupt.

    1.By resetting microprocessor (External signal)
    2.By giving a high TRAP ACKNOWLEDGE (Internal signal)

RST 7.5:

- The RST 7.5 interrupt is a maskable interrupt.

- It has the second highest priority.

- It is edge sensitive. ie. Input goes to high and no need to maintain high state until it recognized.

- Maskable interrupt. It is disabled by,

    1 .DI instruction
    2. System or processor reset.
    3. After reorganization of interrupt.

- Enabled by EI instruction.

RST 6.5 and 5.5:

- The RST 6.5 and RST 5.5 both are level triggered. . ie. Input goes to high and stays high until it recognized.

- Maskable interrupt. It is disabled by,

    1. DI, SIM instruction
    2. System or processor reset.
    3. After reorganization of interrupt.

- Enabled by EI instruction.

- The RST 6.5 has the third priority whereas RST 5.5 has the fourth priority.

INTR:

- INTR is a maskable interrupt. It is disabled by,

    1. DI, SIM instruction
    2. System or processor reset.
    3. After reorganization of interrupt.

- Enabled by EI instruction.

- Non- vectored interrupt. After receiving INTA (active low) signal, it has to supply the address of ISR.

- It has lowest priority.

- It is a level sensitive interrupts. ie. Input goes to high and it is necessary to maintain high state until it recognized.

- The following sequence of events occurs when INTR signal goes high.

1. The 8085 checks the status of INTR signal during execution of each instruction.

2. If INTR signal is high, then 8085 complete its current instruction and sends active low interrupt acknowledge signal, if the interrupt is enabled.

3. In response to the acknowledgement signal, external logic places an instruction OPCODE on the data bus. In the case of multi byte instruction, additional interrupt acknowledge machine cycles are generated by the 8085 to transfer the additional bytes into the microprocessor.

4. On receiving the instruction, the 8085 save the address of next instruction on stack and execute received instruction.

SIM and RIM for interrupts:

- The 8085 provide additional masking facility for RST 7.5, RST 6.5 and RST 5.5 using SIM instruction.

- The status of these interrupts can be read by executing RIM instruction.The masking or unmasking of RST 7.5, RST 6.5 and RST 5.5 interrupts can be performed by moving an 8-bit data to accumulator and then executing SIM instruction.

- The format of the 8-bit data is shown below.

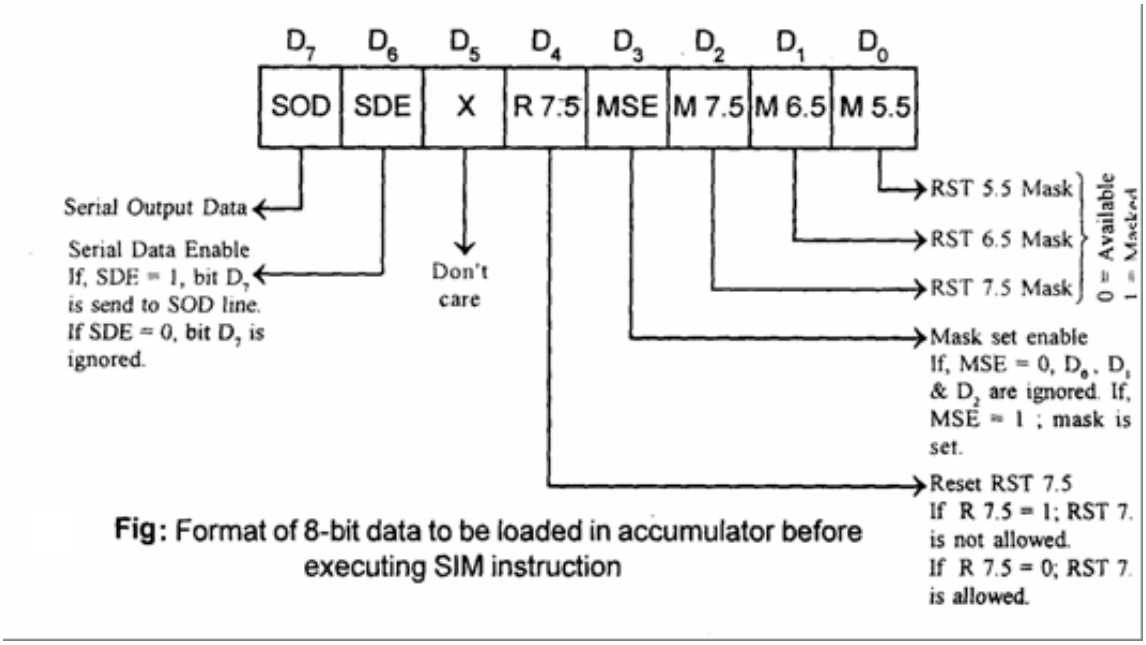
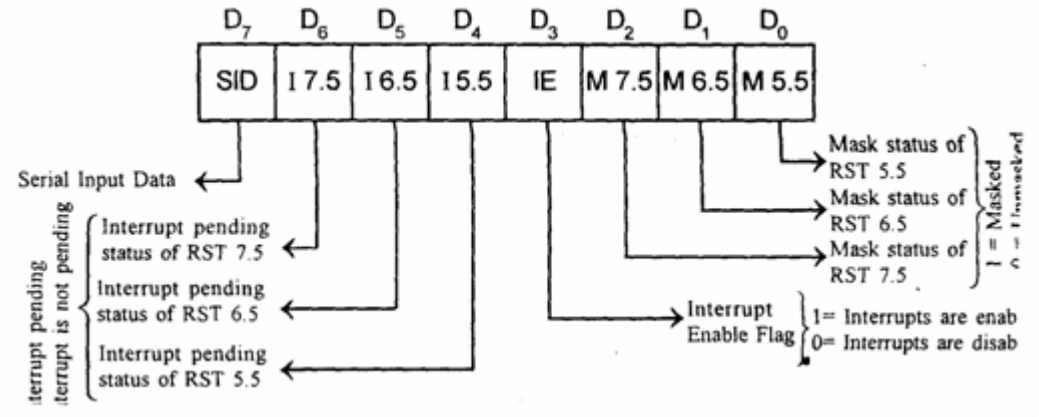


Fig: Format of 8-bit data to be loaded in accumulator before executing SIM instruction

- The status of pending interrupts can be read from accumulator after executing RIM instruction.
- When RIM instruction is executed an 8-bit data is loaded in accumulator, which can be interpreted as shown in fig.



TIMING DIAGRAM

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

Instruction Cycle:

The time required to execute an instruction is called instruction cycle.

Machine Cycle:

The time required to access the memory or input/output devices is called machine cycle.

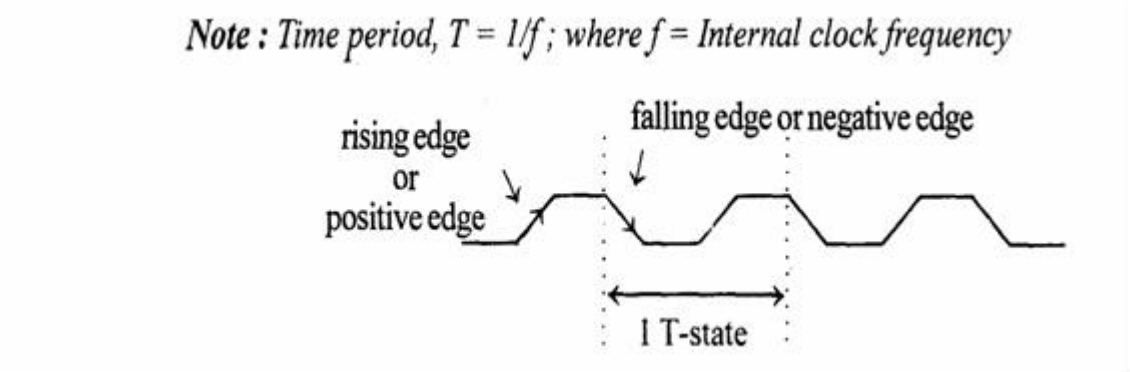
T-State:

- The machine cycle and instruction cycle takes multiple clock periods.
- A portion of an operation carried out in one system clock period is called as T-state.

Machine cycles of 8085:

The 8085 microprocessor has 5 (seven) basic machine cycles. They are

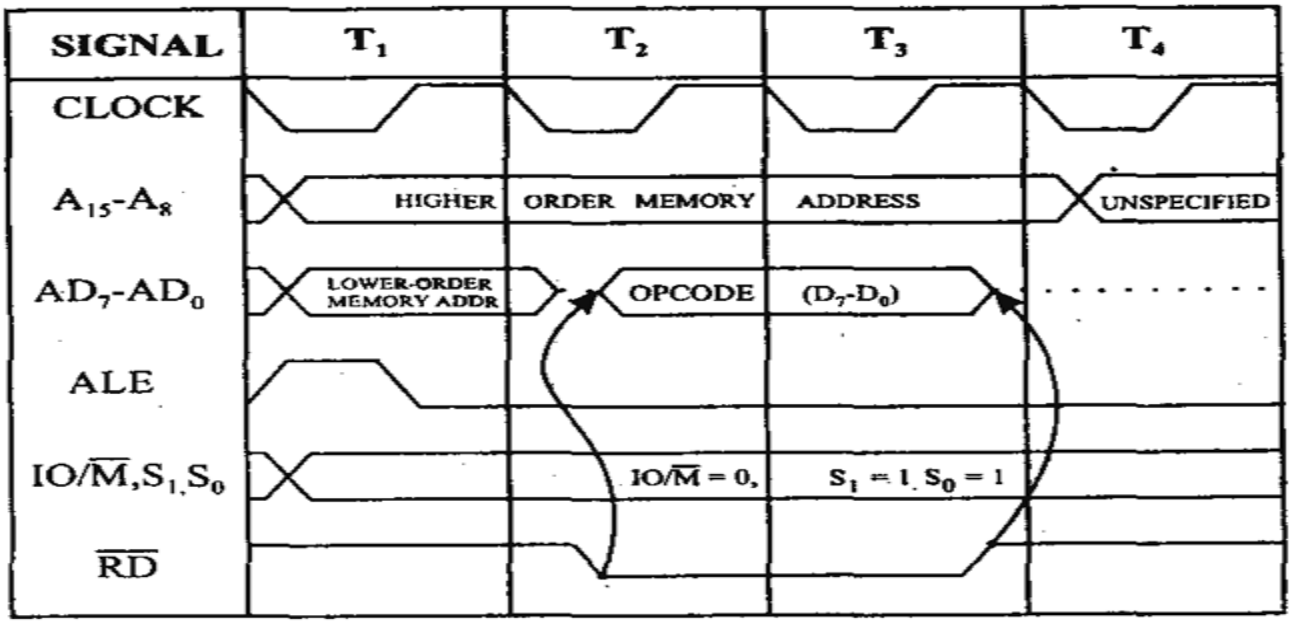
1. Opcode fetch cycle (4T)
2. Memory read cycle (3 T)
3. Memory write cycle (3 T)
4. I/O read cycle (3 T)
5. I/O write cycle (3 T)



- Each instruction of the 8085 processor consists of one to five machine cycles, i.e., when the 8085 processor executes an instruction, it will execute some of the machine cycles in a specific order.
- The processor takes a definite time to execute the machine cycles. The time taken by the processor to execute a machine cycle is expressed in T-states.
- One T-state is equal to the time period of the internal clock signal of the processor.
- The T-state starts at the falling edge of a clock.

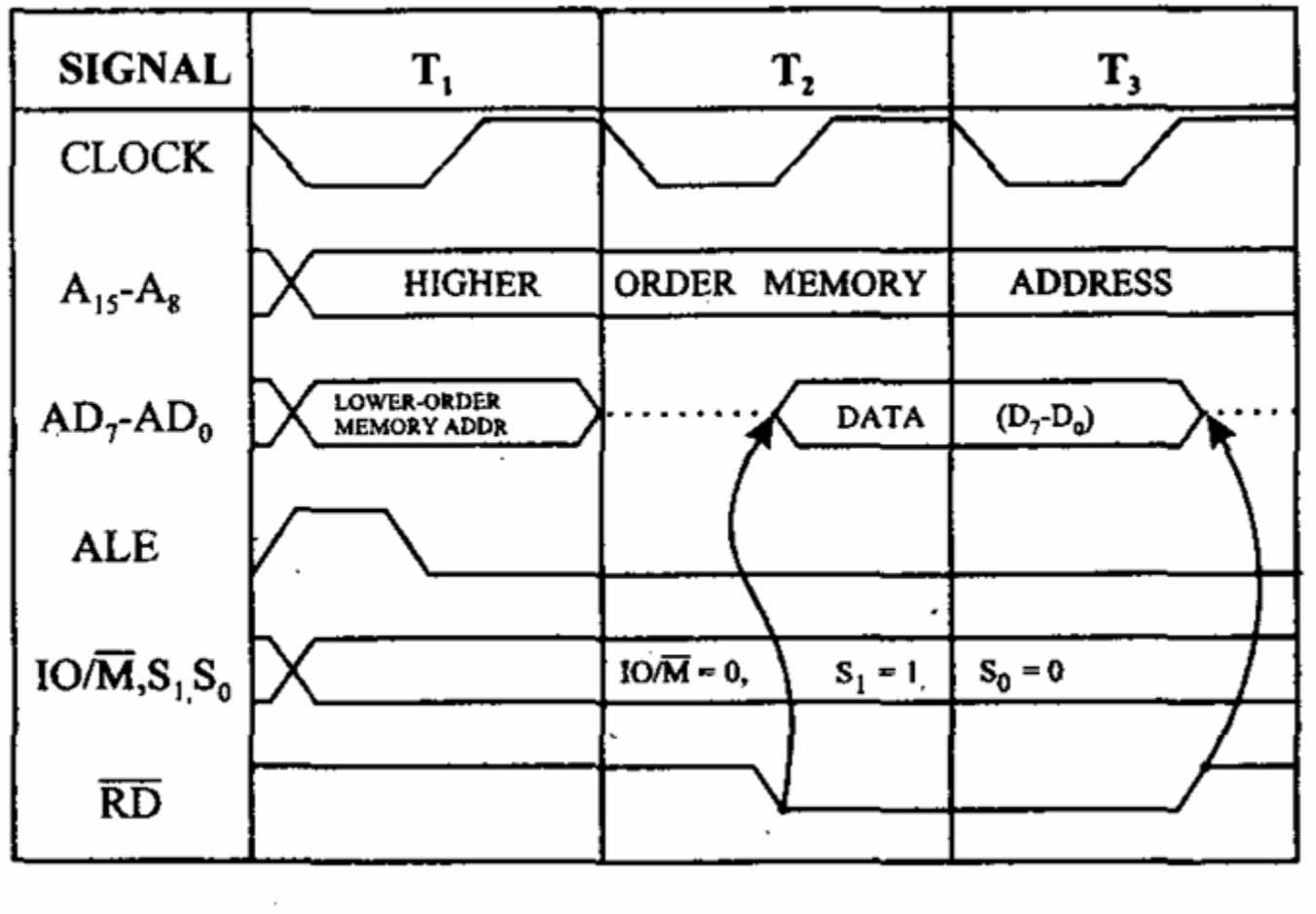
Opcode fetch machine cycle of 8085:

- Each instruction of the processor has one byte opcode.
- The opcode are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.



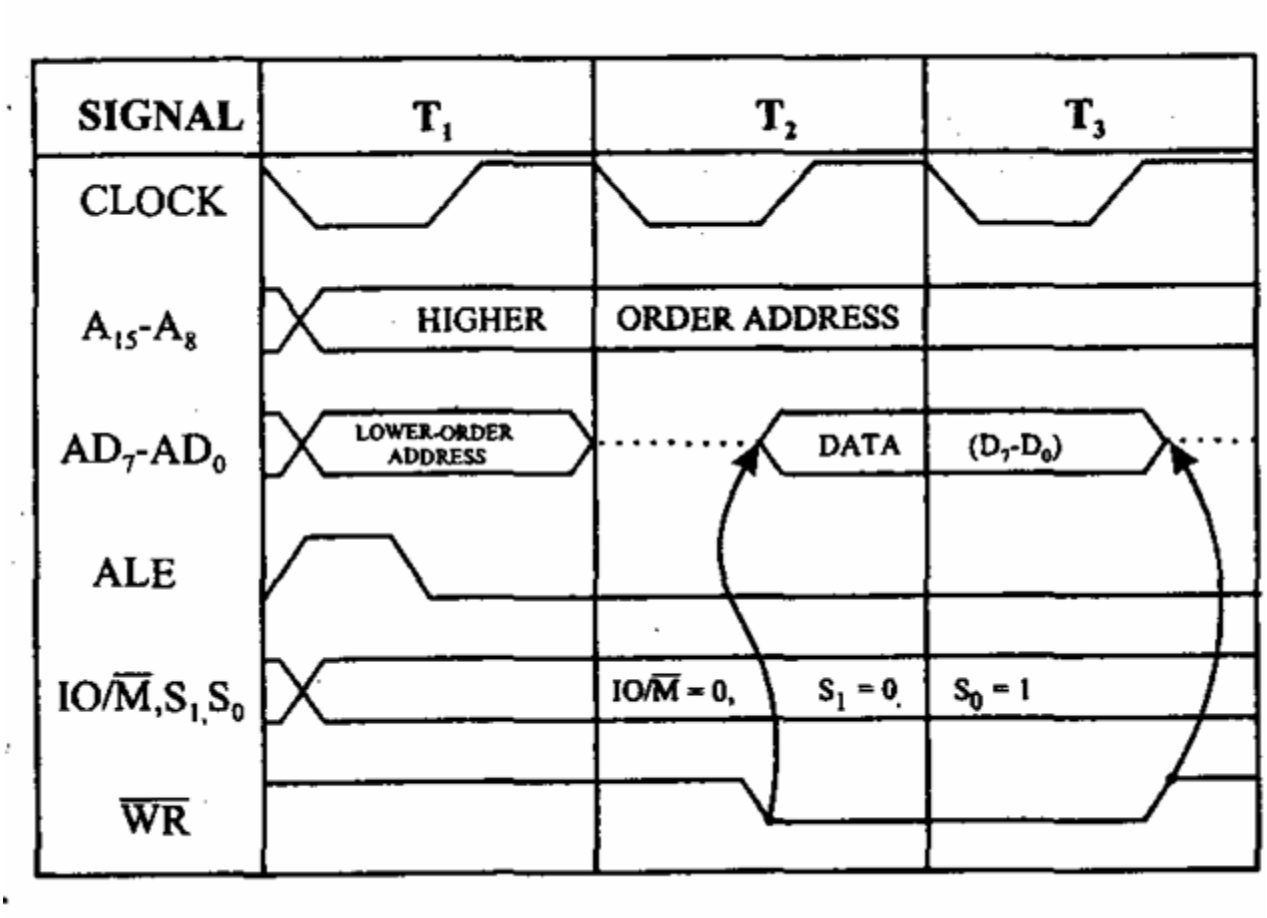
Memory Read Machine Cycle of 8085:

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.



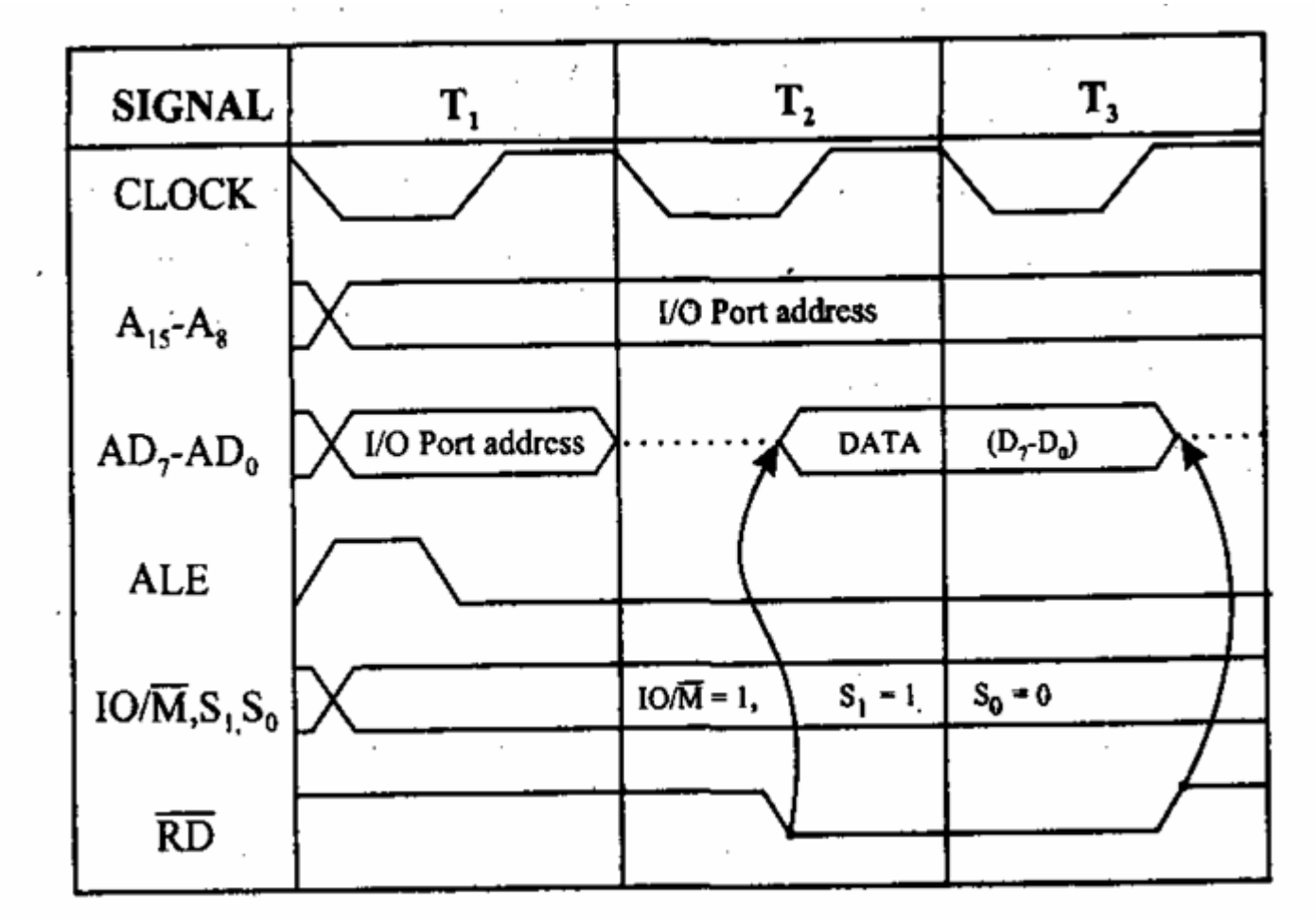
Memory Write Machine Cycle of 8085:

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes 3T states to execute this machine cycle.



I/O Read Cycle of 8085:

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.

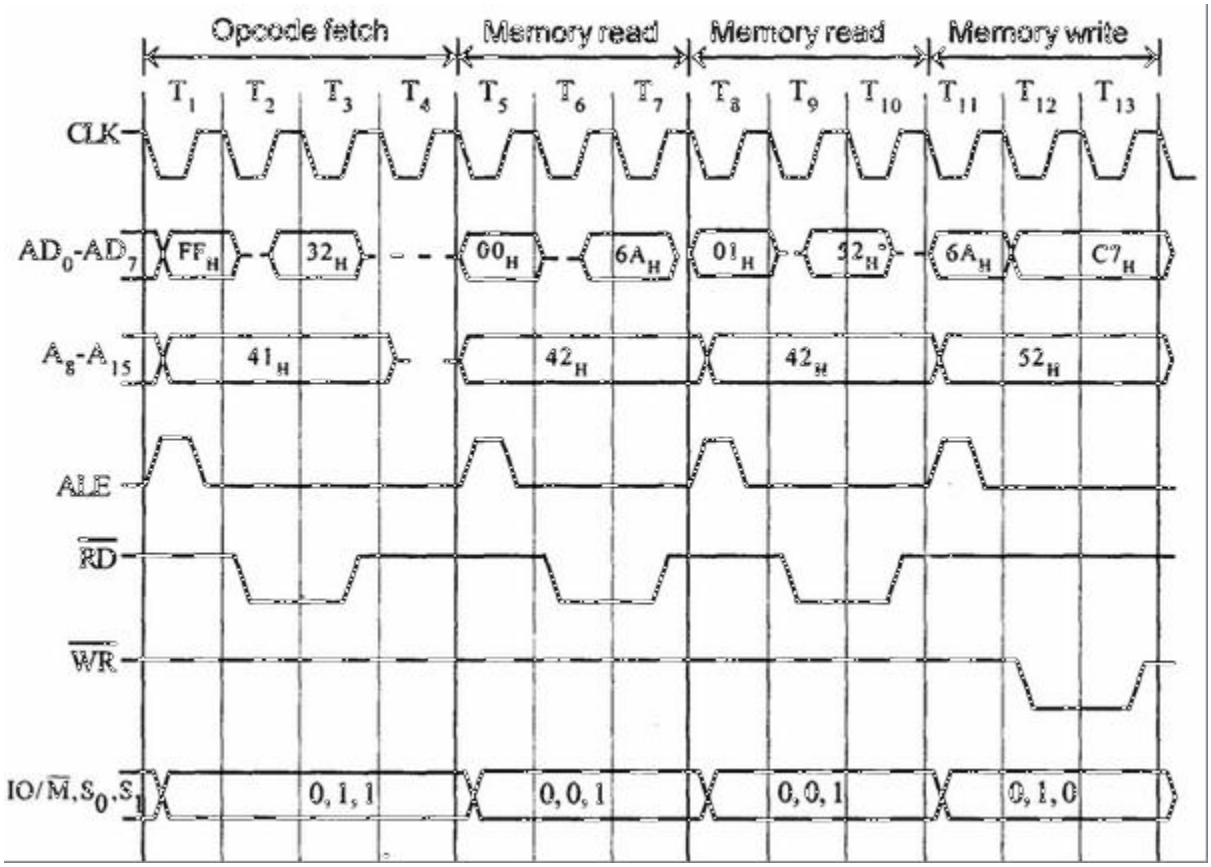


Example

Timing diagram for STA 526AH.

- STA means Store Accumulator -The contents of the accumulator are stored in the specified address (526A).
- The opcode of the STA instruction is said to be 32H. It is fetched from the memory 41FFH (see fig). -*OF machine cycle*
- Then the lower order memory address is read(6A). - *Memory Read Machine Cycle*
- Read the higher order memory address (52).- *Memory Read Machine Cycle*
- The combination of both the addresses is considered and the content from accumulator is written in 526A. - *Memory Write Machine Cycle*
- Assume the memory address for the instruction and let the content of accumulator is C7H. So, C7H from accumulator is now stored in 526A.

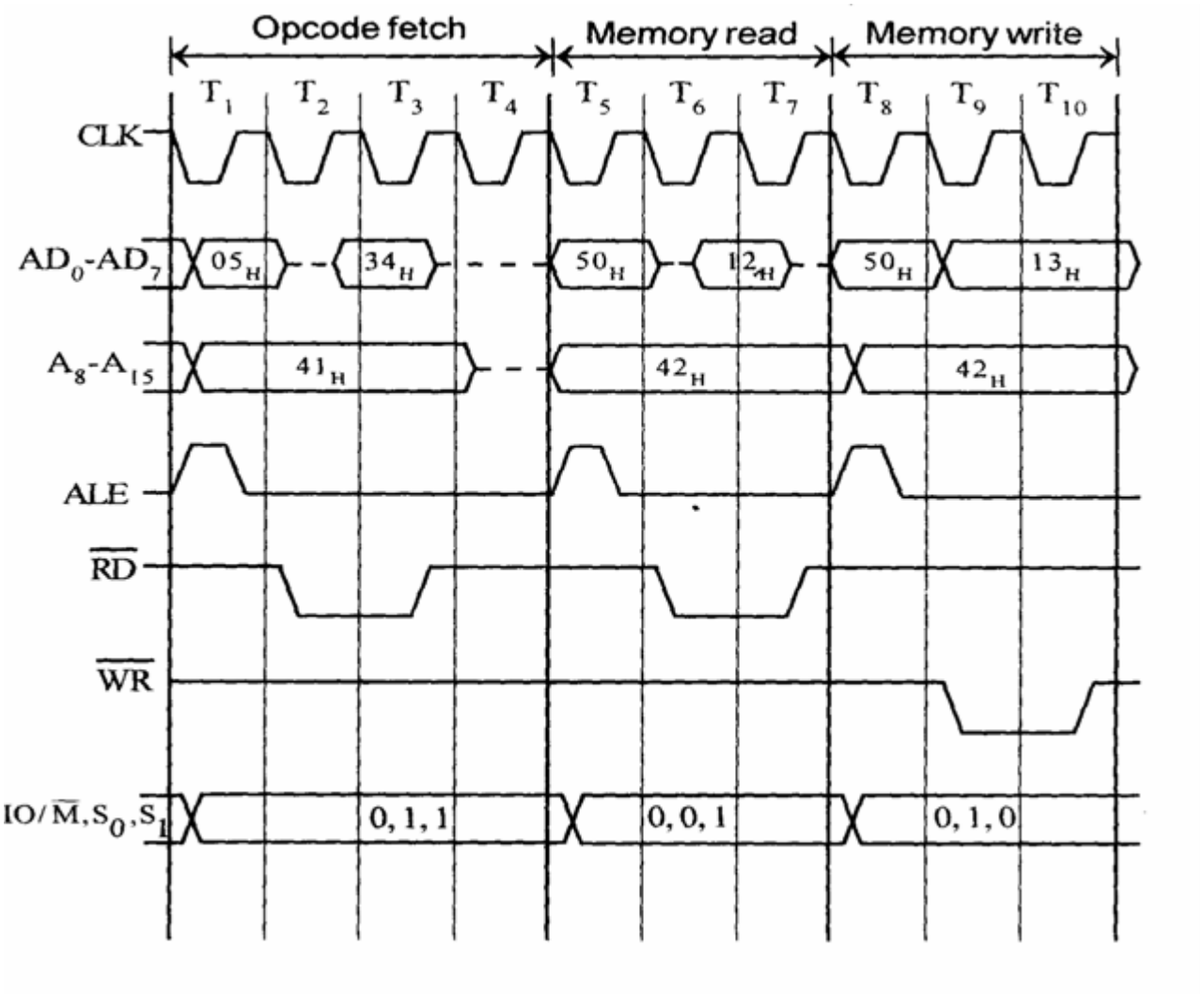
Address	Mnemonics	Opcode
41FF	STA 526A$_H$	32$_H$
4200		6A$_H$
4201		52$_H$



Timing diagram for INR M

- Fetching the Opcode 34H from the memory 4105H. (OF cycle)
- Let the memory address (M) be 4250H. (MR cycle -To read Memory address and data)
- Let the content of that memory is 12H.
- Increment the memory content from 12H to 13H. (MW machine cycle)

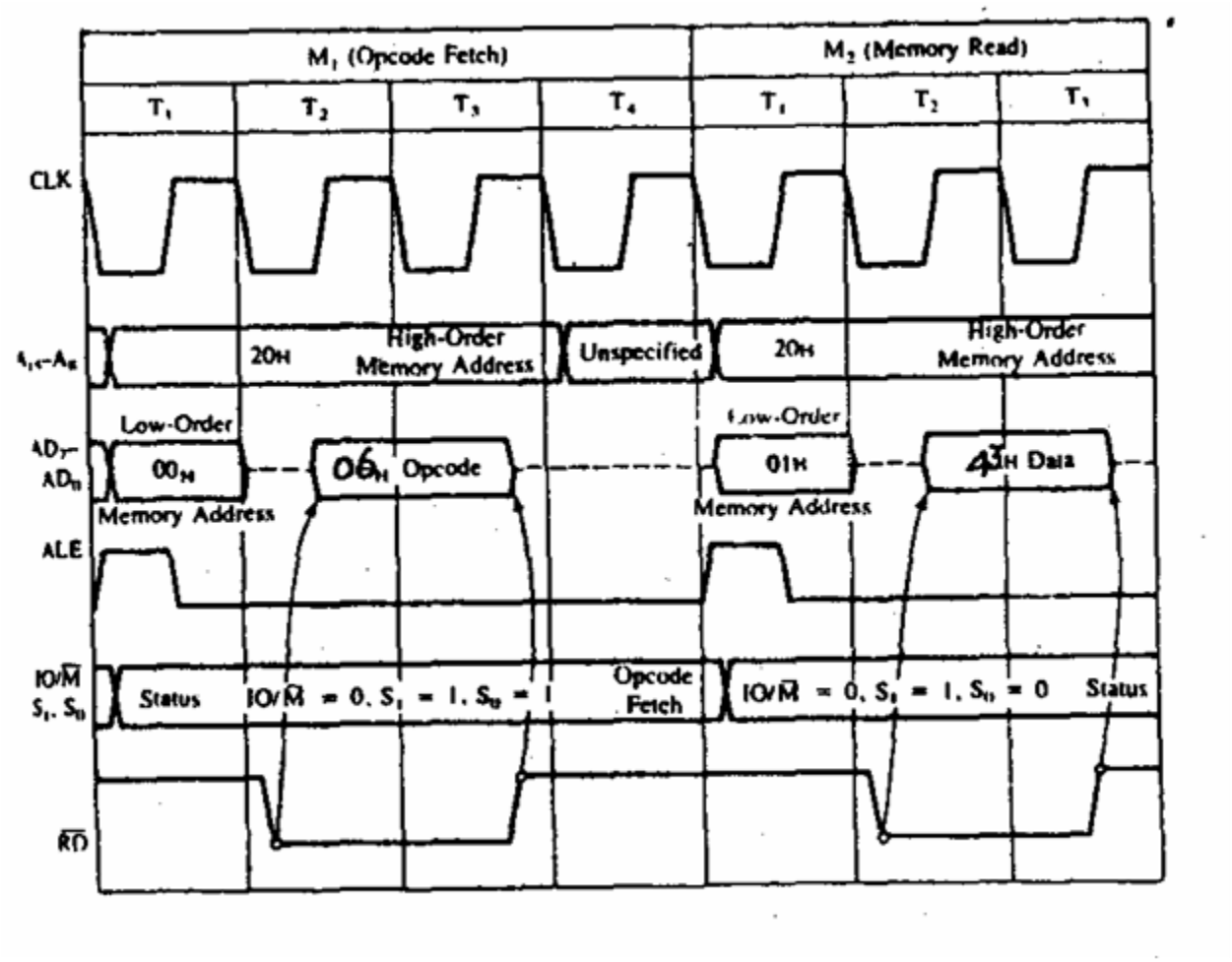
Address	Mnemonics	Opcode
4105	INR M	34$_H$



Timing diagram for MVI B, 43H.

- Fetching the Opcode 06H from the memory 2000H. (OF machine cycle)
- Read (move) the data 43H from memory 2001H. (memory read)

Address	Mnemonics	Opcode
2000	MVI B, 43H	06H
2001		43H


M1 (Opcode Fetch)
M2 (Memory Read)
T1
T2
T3
T4
T1
T2
T3
CLK
A15–A8
20H
High-Order Memory Address
Unspecified
20H
High-Order Memory Address
AD7–AD0
Low-Order
00H
Memory Address
06H Opcode
Low-Order
01H
Memory Address
43H Data
ALE
IO/M
S1, S0
Status
IO/M = 0, S1 = 1, S0 = 1
Opcode Fetch
IO/M = 0, S1 = 1, S0 = 0
Status
RD