Unit 2

Microprocessor architecture and the instruction set

==========

Internal architecture of 8085 microprocessor

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration –

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock
- It is used in washing machines, microwave ovens, mobile phones, etc.

Pin diagram of 8085 Microprocessor

The following image depicts the pin diagram of 8085 Microprocessor.

The pins of an 8085 microprocessor can be classified into *4 groups*:

Group 1:

1. **Address bus**

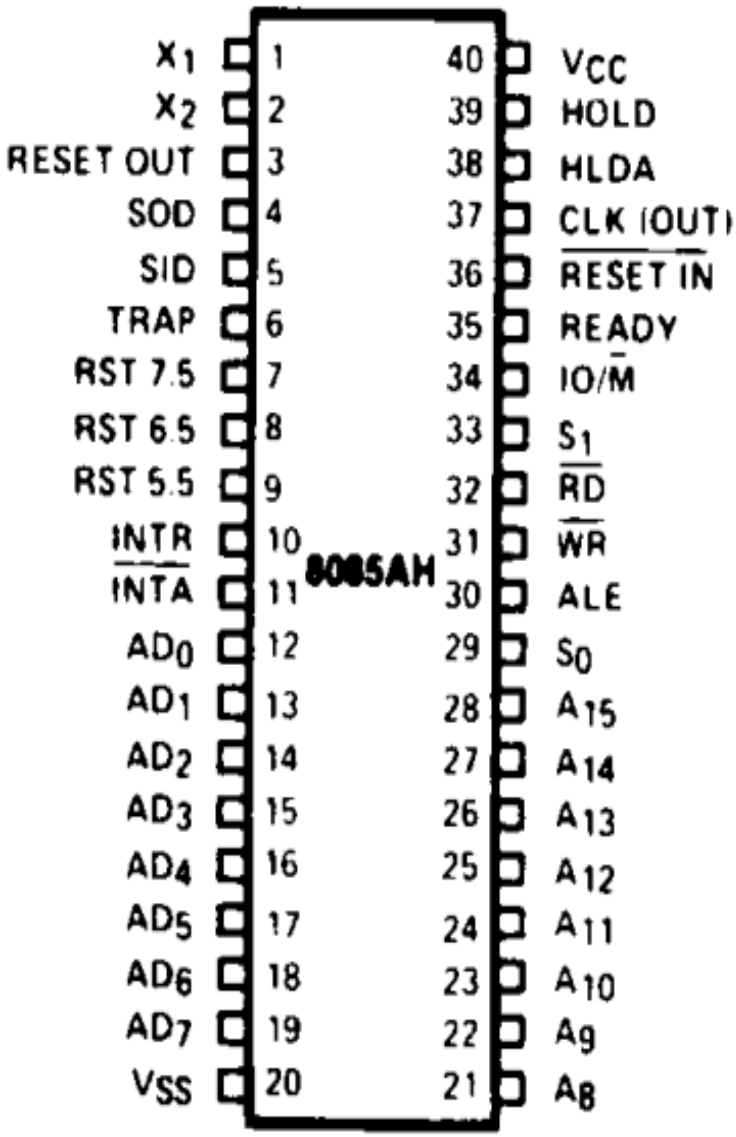
   A15-A8, it carries the most significant 8-bits of memory/IO address.

2. **Data bus**

   AD7-AD0, it carries the least significant 8-bit address and data bus.

3. **Control and status signals**

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.



Three control signals are ~~RD, WR~~ & ALE.

- **RD** – This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** – This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** – It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

4. IO/~~M~~

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

5. S1 & S0

These signals are used to identify the type of current operation.

Group 2:

6. Power supply

There are 2 power supply signals – VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

7. Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** – A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** – This signal is used as the system clock for devices connected with the microprocessor.

Group 3:

8. Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA** – It is an interrupt acknowledgment signal.
- **RESET IN** – This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** – This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** – This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** – This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** – It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

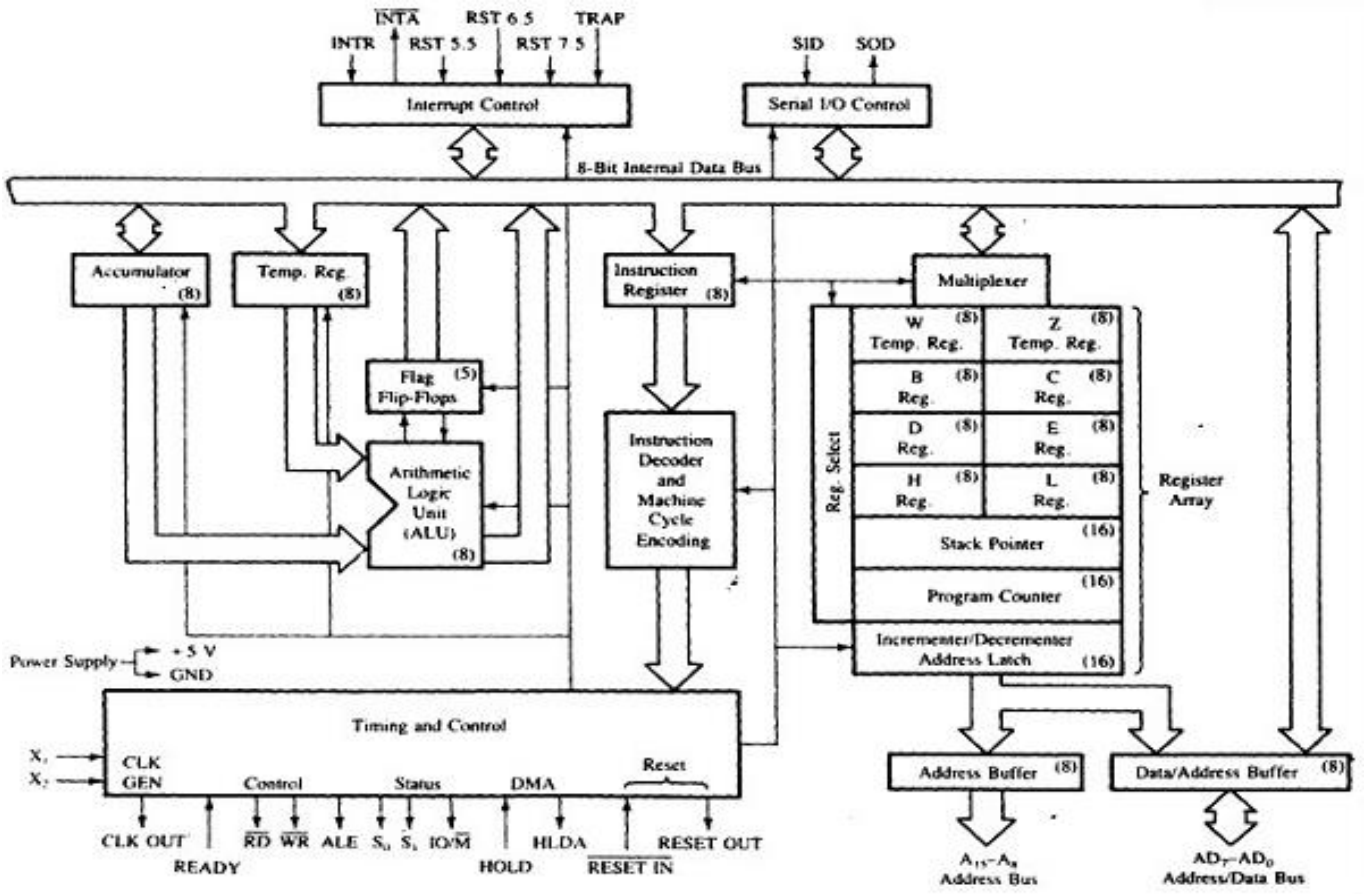
Group 4:

9. Serial I/O signals

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) – The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) – The data on this line is loaded into accumulator whenever a RIM instruction is executed.

Functional Units or Diagram of 8085 MP:



8085 consists of the following functional units −

1. **Accumulator**

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

2. **Arithmetic and logic unit**

   As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

3. **General purpose register**

   There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

   These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

4. Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

5. Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

6. Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

7. Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table –

D7	D6	D5	D4	D3	D2	D1	D0
S	**Z**	**-**	**AC**	**-**	**P**	**-**	**CY**

8. Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

9. Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

10.Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessors: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

11.Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

12.Address buffer and address-data buffer

The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

13.Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

8085 Addressing Modes & Interrupts

The way of specifying data to be operated by an instruction is called addressing mode.

Types of addressing modes –

In 8085 microprocessor there are 5 types of addressing modes:

1. **Immediate Addressing Mode –**
   In immediate addressing mode the source operand is always data. If the data is 8-bit, then the instruction will be of 2 bytes, if the data is of 16-bit then the instruction will be of 3 bytes.

   Examples:
   MVI B 45 (move the data 45H immediately to register B)
   LXI H 3050 (load the H-L pair with the operand 3050H immediately)
   JMP address (jump to the operand address immediately)

2. **Register Addressing Mode –**
   In register addressing mode, the data to be operated is available inside the register(s) and register(s) is(are) operands. Therefore, the operation is performed within various registers of the microprocessor.

   Examples:
   MOV A, B (move the contents of register B to register A)
   ADD B (add contents of registers A and B and store the result in register A)
   INR A (increment the contents of register A by one)

3. **Direct Addressing Mode –**
   In direct addressing mode, the data to be operated is available inside a memory location and that memory location is directly specified as an operand. The operand is directly available in the instruction itself.

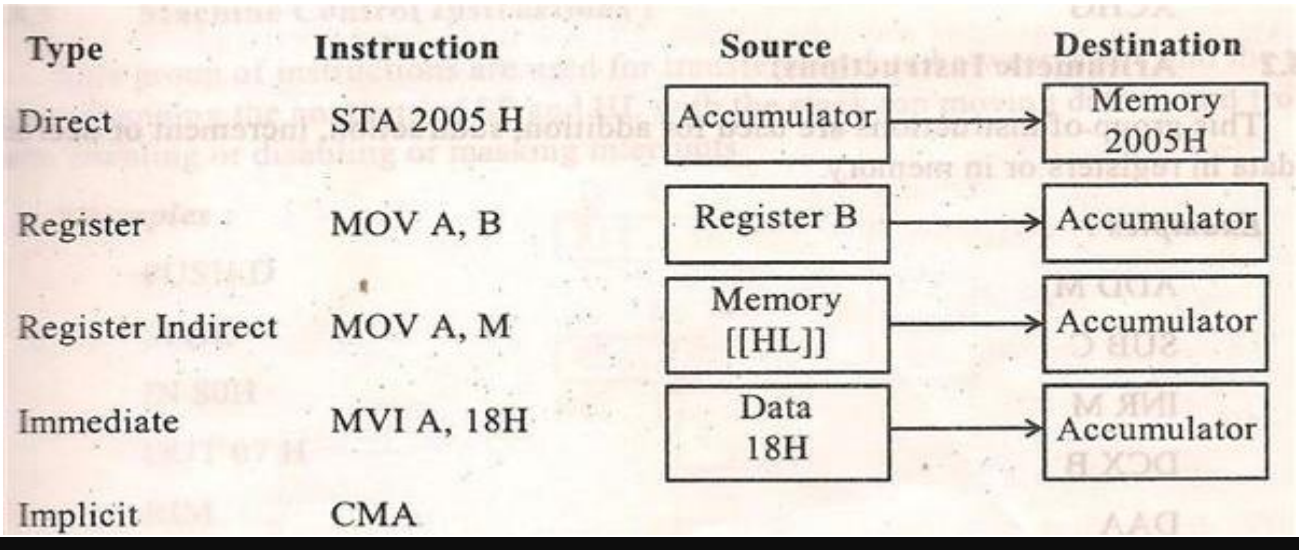
   Examples:
   LDA 2050 (load the contents of memory location into accumulator A)
   LHLD address (load contents of 16-bit memory location into H-L register pair)
   IN 35 (read the data from port whose address is 01)

4. **Register Indirect Addressing Mode –**
   IN register indirect addressing mode, the data to be operated is available inside a memory location and that memory location is indirectly specified b a register pair.

   Examples:
   MOV A, M (move the contents of the memory location pointed by the H-L pair to the accumulator)
   LDAX B (move contains of B-C register to the accumulator)
   LXIH 9570 (load immediate the H-L pair with the address of the location 9570)

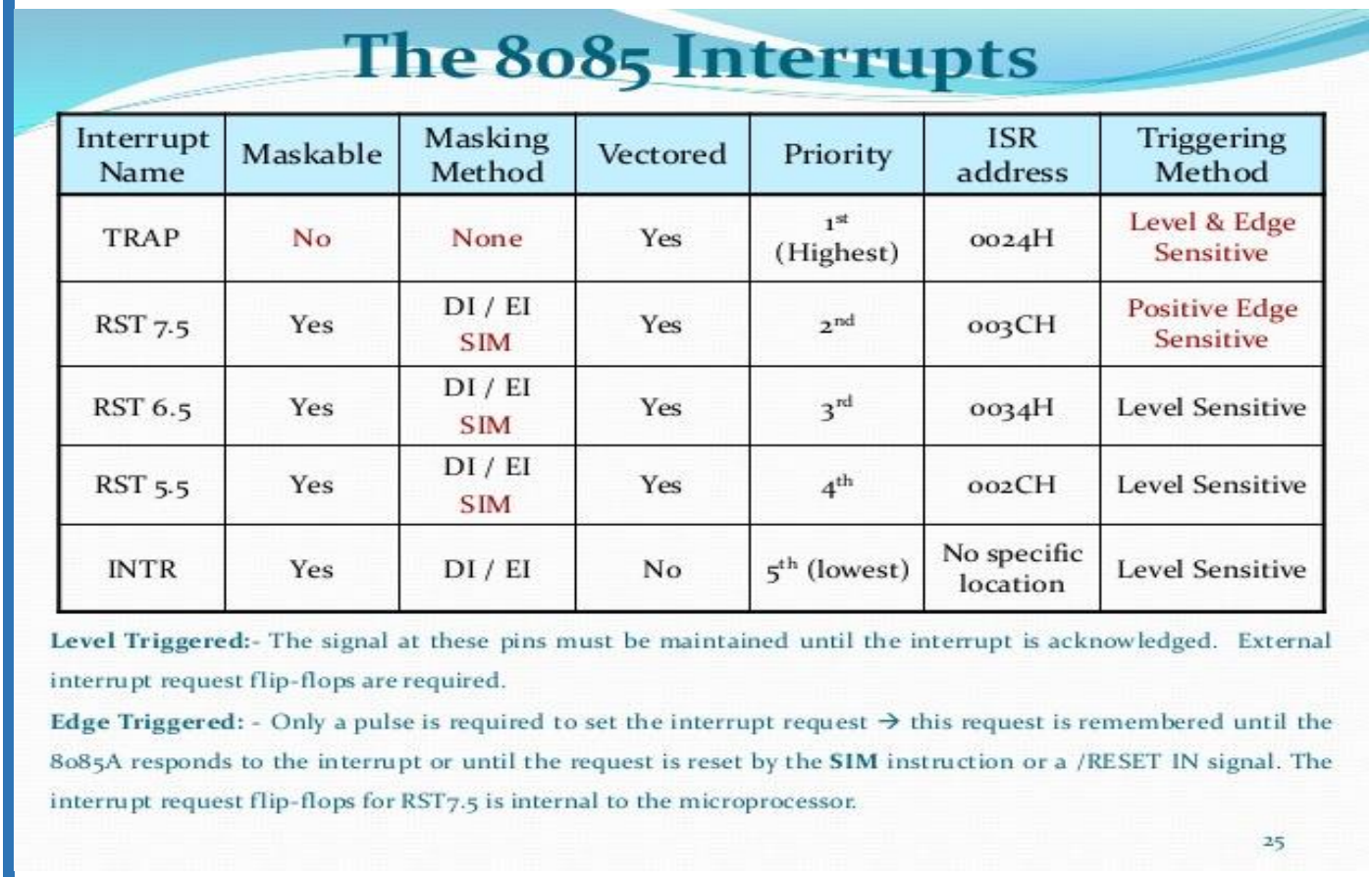
5. **Implied/Implicit Addressing Mode –**
   In implied/implicit addressing mode the operand is hidden and the data to be operated is available in the instruction itself.

   Examples:
   CMA (finds and stores the 1's complement of the contains of accumulator A in A)
   RRC (rotate accumulator A right by one bit)
   RLC (rotate accumulator A left by one bit)



Interrupts in 8085

Interrupts are the signals generated by the external devices to request the microprocessor to perform a task. There are **5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.**



The 8085 Interrupts

Interrupt Name	Maskable	Masking Method	Vectored	Priority	ISR address	Triggering Method
TRAP	No	None	Yes	1st (Highest)	0024H	Level & Edge Sensitive
RST 7.5	Yes	DI / EI SIM	Yes	2nd	003CH	Positive Edge Sensitive
RST 6.5	Yes	DI / EI SIM	Yes	3rd	0034H	Level Sensitive
RST 5.5	Yes	DI / EI SIM	Yes	4th	002CH	Level Sensitive
INTR	Yes	DI / EI	No	5th (lowest)	No specific location	Level Sensitive

Level Triggered:- The signal at these pins must be maintained until the interrupt is acknowledged. External interrupt request flip-flops are required.

Edge Triggered: - Only a pulse is required to set the interrupt request → this request is remembered until the 8085A responds to the interrupt or until the request is reset by the **SIM** instruction or a /RESET IN signal. The interrupt request flip-flops for RST7.5 is internal to the microprocessor.

25

Interrupt are classified into following groups based on their parameter −

- **Vector interrupt** − In this type of interrupt, the interrupt address is known to the processor. **For example:** RST7.5, RST6.5, RST5.5, TRAP.
- **Non-Vector interrupt** − In this type of interrupt, the interrupt address is not known to the processor so, the interrupt address needs to be sent externally by the device to perform interrupts. **For example:** INTR.
- **Maskable interrupt** − In this type of interrupt, we can disable the interrupt by writing some instructions into the program. **For example:** RST7.5, RST6.5, RST5.5.
- **Non-Maskable interrupt** − In this type of interrupt, we cannot disable the interrupt by writing some instructions into the program. **For example:** TRAP.

- **Software interrupt** – In this type of interrupt, the programmer has to add the instructions into the program to execute the interrupt. There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7.
- **Hardware interrupt** – There are 5 interrupt pins in 8085 used as hardware interrupts, i.e. TRAP, RST7.5, RST6.5, RST5.5, INTA.

Note – NTA is not an interrupt, it is used by the microprocessor for sending acknowledgement. TRAP has the highest priority, then RST7.5 and so on.

Interrupt Service Routine (ISR)

A small program or a routine that when executed, services the corresponding interrupting source is called an ISR.

TRAP

It is a non-maskable interrupt, having the highest priority among all interrupts. Bydefault, it is enabled until it gets acknowledged. In case of failure, it executes as ISR and sends the data to backup memory. This interrupt transfers the control to the location 0024H.

RST7.5

It is a maskable interrupt, having the second highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 003CH address.

RST 6.5

It is a maskable interrupt, having the third highest priority among all interrupts. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 0034H address.

RST 5.5

It is a maskable interrupt. When this interrupt is executed, the processor saves the content of the PC register into the stack and branches to 002CH address.

INTR

It is a maskable interrupt, having the lowest priority among all interrupts. It can be disabled by resetting the microprocessor.

When **INTR signal goes high**, the following events can occur –

1. The microprocessor checks the status of INTR signal during the execution of each instruction.
2. When the INTR signal is high, then the microprocessor completes its current instruction and sends active low interrupt acknowledge signal.
3. When instructions are received, then the microprocessor saves the address of the next instruction on stack and executes the received instruction.

Instruction and data formats

It Consists of:

- ✓ One-word or 1-byte instructions
- ✓ Two-word or 2-byte instructions
- ✓ Three-word or 3-byte instructions

Primary acknowledgement:

Instruction: *It is command given to a computer to perform specific operation on given data. The entire group of instructions called the instruction set determines what functions the microprocessor can perform. It has two parts*

Opcode: *Specifies what operation to be performed.*

Operand: *Specifies where to perform the operation.*

Opcode	***Operand***

The computer can be used to perform a specific task only by specifying the necessary steps needed to complete the task. The collection of such ordered steps

*forms a program of a computer. The program is thus collection form of such steps called **instruction**.*

The 8085A instruction set consists of one, two and three byte instructions.

The first byte is always the opcode; in two-byte instructions the second byte is usually data; in three byte instructions the last two bytes' present address or 16-bit data.

1. **One-byte instruction:**

For Example: MOV A, B whose opcode is 78H which is one byte. This Instruction and Data Format of 8085 copies the contents of B register in A register.

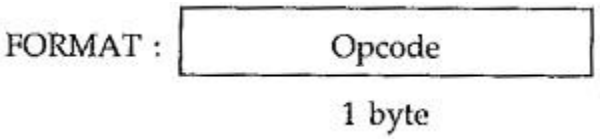


Table 2.1 Example for 1 byte Instruction

Task	Op code	Operand	Binary Code	Hex Code
Copy the contents of the accumulator in the register C.	MOV	C,A	0100 1111	4FH
Add the contents of register B to the contents of the accumulator.	ADD	B	1000 0000	80H
Invert (compliment) each bit in the accumulator.	CMA		0010 1111	2FH

2. Two-byte instruction:

For Example: MVI B, 02H. The opcode for this instruction is 06H and is always followed by a byte data (02H in this case). This instruction is a two-byte instruction which copies immediate data into B register.

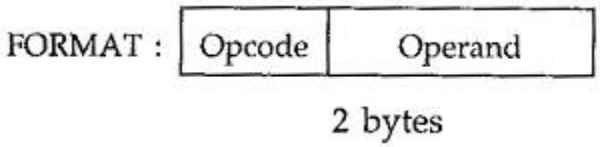


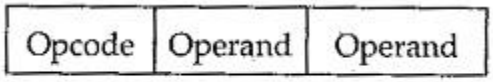
Table 2.2 Example for 2 byte Instruction

Task	Opcode	Operand	Binary Code	Hex Code	
Load an 8-bit data byte in the accumulator.	MVI	A, Data	0011 1110 DATA	3E Data	First Byte Second Byte

***3.* Three-byte instruction:**

For Example: JMP 6200H. The opcode for this instruction is C3H and is always followed by 16-bit address (6200H in this case). This instruction is a three-byte instruction which loads 16-bit address into program counter.

FORMAT :



3 bytes

Table 3.3 Example for 3 byte Instruction

Task	Opcode	Operand	Binary code	Hex Code	
Transfer the program sequence to the memory location 2085H.	JMP	2085H	1100 0011 1000 0101 0010 0000	C3 85 20	First byte Second Byte Third Byte

Classification of Instruction Sets

The instruction sets can be differentiated by

- ✓ Operand storage in the CPU
- ✓ Number of explicit operands per instruction
- ✓ Operand location
- ✓ Operations
- ✓ Type and size of operands

The type of internal storage in the CPU is the most basic differentiation. The major choices are

- ✓ **a stack** (the operands are implicitly on top of the stack)
- ✓ **an accumulator** (one operand is implicitly the accumulator)
- ✓ **a set of registers** (all operands are explicit either registers or memory locations)

The code segment C = A + B how it would appear on the classes of instruction sets

Stack	**Accumulator**	**Register**
PUSH A	Load A	Load R1,A
PUSH B	ADD B	ADD R1,B
ADD	Store C	Store C,R1
POP C		

The code segment C = A + B how it would appear on the classes of instruction sets

Stack	**Accumulator**	**Register**
PUSH A	Load A	Load R1,A
PUSH B	ADD B	ADD R1,B
ADD	Store C	Store C,R1
POP C		

While most early machines used stack or accumulator-style architectures, all machines designed in the past ten years use a general purpose architecture. The reason is the registers are:

- ✓ faster than memory
- ✓ easier for a compiler to use
- ✓ can be used more effectively

Primary advantages and disadvantages of each class of machine

Machine Type	**Advantages**	**Disadvantages**
Stack	Simple model of expression evaluation. Good code density.	A stack can't be randomly accessed. It makes it difficult to generate efficient code.
Accumulator	Minimizes internal state of machine. Short instructions	Since accumulator is only temporary storage, memory traffic is highest.

Register	Most general model for code generation	All operands must be named, leading to longer instructions.

Classification of General Purpose Register Machines

There are two major instruction set characteristics that divide GPR architectures. They concern

1. whether an ALU instruction has two or three operands

ADD R3, R1, R2
R3 <-R1 + R2

Or

ADD R1, R2
R1 <- R1 + R2

2. how many of the operands may be memory addressed in ALU instruction

- Register- Register (Load/Store)
  ADD R3, R1, R2 (R3 <- R1 + R2)
- Register - Memory
  ADD R1, A (R1 <- R1 + A)
- Memory - Memory
  ADD C, A, B (C <- A + B)

Instruction Set 8085

1. **Data Transfer Instructions**
2. **Arithmetic Instruction**
3. **Logical Instruction**
4. **Branching Instruction**
5. **Control Instruction (or IO & Machine control Instruction)**

1. Data Transfer Instructions

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 or MOV R1,R2
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. Example: 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 high order 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

2.Arithmetic Instructions

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. Example: 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

3.LOGICAL INSTRUCTIONS

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	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 with accumulator	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 data**	Exclusive OR immediate with accumulator	The contents of the accumulator are Exclusive 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: 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

4.Branching Instruction

<table>
<tr><th>Opcode</th><th>Operand</th><th>Explanation of Instruction</th><th>Description</th></tr>
<tr><td>JMP</td><td>16-bit address</td><td>Jump unconditionally</td><td>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</td></tr>
<tr><td><table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>JC</td><td>Jump on Carry</td><td>CY = 1</td></tr><tr><td>JNC</td><td>Jump on no Carry</td><td>CY = 0</td></tr><tr><td>JP</td><td>Jump on positive</td><td>S = 0</td></tr><tr><td>JM</td><td>Jump on minus</td><td>S = 1</td></tr><tr><td>JZ</td><td>Jump on zero</td><td>Z = 1</td></tr><tr><td>JNZ</td><td>Jump on no zero</td><td>Z = 0</td></tr><tr><td>JPE</td><td>Jump on parity even</td><td>P = 1</td></tr><tr><td>JPO</td><td>Jump on parity odd</td><td>P = 0</td></tr></table></td><td>16-bit address</td><td>Jump conditionally</td><td>The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.

Example: JZ 2034H or JZ XYZ</td></tr>
<tr><td><table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>CC</td><td>Call on Carry</td><td>CY = 1</td></tr><tr><td>CNC</td><td>Call on no Carry</td><td>CY = 0</td></tr><tr><td>CP</td><td>Call on positive</td><td>S = 0</td></tr><tr><td>CM</td><td>Call on minus</td><td>S = 1</td></tr><tr><td>CZ</td><td>Call on zero</td><td>Z = 1</td></tr><tr><td>CNZ</td><td>Call on no zero</td><td>Z = 0</td></tr><tr><td>CPE</td><td>Call on parity even</td><td>P = 1</td></tr><tr><td>CPO</td><td>Call on parity odd</td><td>P = 0</td></tr></table></td><td>16-bit address</td><td>Unconditional subroutine call</td><td>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</td></tr>
<tr><td>RET</td><td>none</td><td>Return from subroutine unconditionally</td><td>The program sequence is transferred from the subroutine to the calling</td></tr>
</table>

			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
<table><tr><th>Opcode</th><th>Description</th><th>Flag Status</th></tr><tr><td>RC</td><td>Return on Carry</td><td>CY = 1</td></tr><tr><td>RNC</td><td>Return on no Carry</td><td>CY = 0</td></tr><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>	**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 copied into the program counter, and program execution begins at the new address. Example: RZ
PCHL	**none**	Load program counter with HL contents	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
RST	**0-7**	Restart	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:

Instruction	Restart Address
RST 0	0000H
RST1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
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

5.Control Instructions (IO Machine Control)

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

Example of Assembly Level Programming(ALP)

Store 8-bit data in memory

```
Program 1:
MVI A, 52H : "Store 32H in the accumulator"
STA 4000H  : "Copy accumulator contents at address 4000H"
HLT        : "Terminate program execution"
```

```
Program 2:
LXI H  : "Load HL with 4000H"
MVI M  : "Store 32H in memory location pointed by HL register pair (4000H)"
HLT    : "Terminate program execution"
```

`Note:` The result of both programs will be the same. In program 1 direct addressing instruction is used, where program 2 indirect addressing instruction is used.

Add two 8-bit numbers

Statement: Add the contents of memory locations 4000H and 4001H and place the result in memory location 4002

```
Sample problem
(4000H) = 14H
(4001H) = 89H
Result     = 14H + 89H = 9DH

Source program
LXI H 4000H : "HL points 4000H"
MOV A, M    : "Get first operand"
INX H       : "HL points 4001H"
ADD M       : "Add second operand"
INX H       : "HL points 4002H"
MOV M, A    : "Store result at 4002H"
HLT         : "Terminate program execution"
```

Subtract two 8-bit numbers

Statement: Subtract the contents of memory location 4001H from the memory location 2000H and place the result

memory location 4002H.

```
Program –: Subtract two 8-bit numbers
Sample problem:
(4000H) = 51H
(4001H) = 19H
Result    = 51H – 19H = 38H

Source program:
LXI H, 4000H : "HL points 4000H"
MOV A, M     : "Get first operand"
INX H        : "HL points 4001H"
SUB M        : "Subtract second operand"
INX H        : "HL points 4002H"
MOV M, A     : "Store result at 4002H"
HLT          : "Terminate program execution"
```

Registers of 8085 microprocessor

- A **microprocessor** is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provide results as output.
- A 8085 microprocessor, is a second generation 8-bit microprocessor and is the base for studying and using all the microprocessor available in the market.

Registers in 8085:

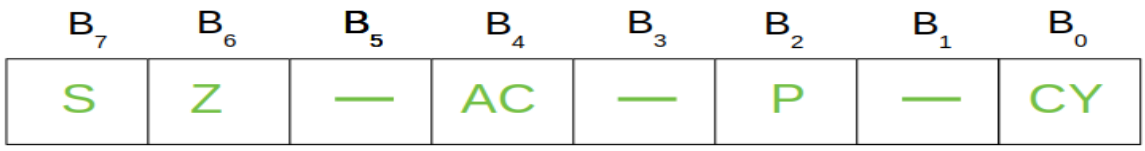
(a) General Purpose Registers –

The 8085 has six general-purpose registers to store 8-bit data; these are identified as- B, C, D, E, H, and L. These can be combined as register pairs – **BC, DE, and HL, to perform some 16-bit operation**. These registers are used to **store or copy** temporary data, by using instructions, during the execution of the program.

(b) Specific Purpose Registers –

- **Accumulator:**
  The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU). After performing arithmetical or logical operations, the result is stored in accumulator. Accumulator is also defined as register A.

- **Flag registers:**



fig(a)-Bit position of various flags in flag registers of 8085

The flag register is a special purpose register and it is completely different from other registers in microprocessor. It consists of **8 bits and only 5 of them are useful**. The other three are left vacant and are used in **the future Intel versions**.

These 5 flags are set or reset (when value of flag is 1, then it is said to be set and when value is 0, then it is said to be reset) after an operation according to data condition of the result in the accumulator and other registers. The 5 flag registers are:

1. Sign Flag:

   It occupies the seventh bit of the flag register, which is also known as the most significant bit.

   It helps the programmer to know whether the number stored in the accumulator is positive or negative.

   If the sign flag is set, it means that number stored in the accumulator is negative, and if reset, then the number is positive.

2. Zero Flag:

   It occupies the sixth bit of the flag register. It is set, when the operation performed in the ALU results in zero(all 8 bits are zero), otherwise it is reset.

   It helps in determining if two numbers are equal or not.

3. Auxillary Carry Flag:

   It occupies the fourth bit of the flag register.

   In an arithmetic operation, when a carry flag is generated by the third bit and passed on to the fourth bit, then Auxillary Carry flag is set. If not flag is reset.

   This flag is used internally for BCD(Binary-Coded decimal Number) operations.

   Note – This is the only flag register in 8085 which is not accessible by user.

4. Parity FlagL:

   It occupies the second bit of the flag register. This flag tests for number of 1's in the accumulator.

   If the accumulator holds even number of 1's, then this flag is set and it is said to even parity. On the other hand if the number of 1's is odd, then it is reset and it is said to be odd parity.

5. Carry Flag:

   It occupies the zeroth bit of the flag register.

   If the arithmetic operation results in a carry(if result is more than 8 bit), then Carry Flag is set; otherwise it is reset.

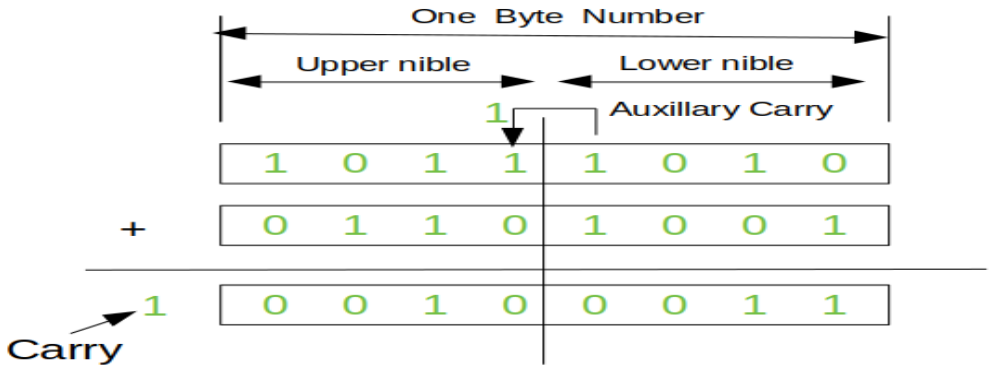
(c) Memory Registers –

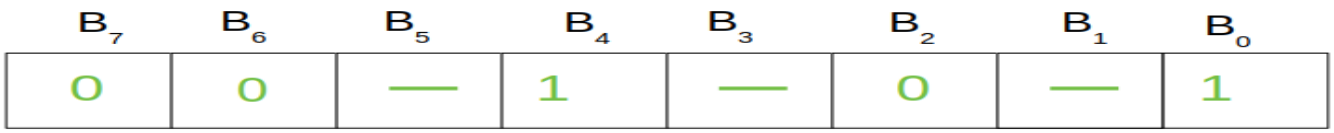
There are two 16-bit registers used to hold memory addresses. The size of these registers is 16 bits because the memory addresses are 16 bits. They are:-

- **Program Counter:** This register is used to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.
- **Stack Pointer:** It is used as a memory pointer. It points to a memory location in read/write memory, called the stack. It is always incremented/decremented by 2 during push and pop operation.

Example –

Here two **binary numbers are added**. The result produced is stored in the accumulator. Now lets check what each bit means. Refer to the below explanation simultaneously to connect them with the example.





Value of flags

- **Sign Flag (7th bit):** It is reset(0), which means number stored in the accumulator is positive.
- **Zero Flag (6th bit):** It is reset(0), thus result of the operations performed in the ALU is non-zero.
- **Auxiliary Carry Flag (4th bit):** We can see that b3 generates a carry which is taken by b4, thus auxiliary carry flag gets set (1).
- **Parity Flag (2nd bit):** It is reset(0), it means that parity is odd. The accumulator holds odd number of 1's.
- **Carry Flag (0th bit):** It is set(1), output results in more than 8 bit.

Example basis on Flag register in 8085 microprocessor

The **Flag register** is a Special Purpose Register. Depending upon the value of result after any arithmetic and logical operation the flag bits become **set (1) or reset (0).**

In 8085 microprocessor, flag register consists of 8 bits and only 5 of them are useful.
The 5 flags are:

1. **Sign Flag (S) –** After any operation if the MSB (B(7)) of the result is 1, it indicates the number is negative and the sign flag becomes set, i.e. 1. If the MSB is 0, it indicates the number is positive and the sign flag becomes reset i.e. 0.
   from 00H to 7F, sign flag is 0
   from 80H to FF, sign flag is 1
   1- MSB is 1 (negative)
   0- MSB is 0 (positive)

   Example:

   MVI A 30 (load 30H in register A)
   MVI B 40 (load 40H in register B)
   SUB B (A = A – B)
   These set of instructions will set the sign flag to 1 as 30 – 40 is a negative number.

MVI A 40 (load 40H in register A)
MVI B 30 (load 30H in register B)
SUB B (A = A – B)
These set of instructions will reset the sign flag to 0 as 40 – 30 is a positive number.

2. Zero Flag (Z) – After any arithmetical or logical operation if the result is 0 (00)H, the zero flag becomes set i.e. 1, otherwise it becomes reset i.e. 0.
   00H zero flag is 1.
   from 01H to FFH zero flag is 0
   1- zero result
   0- non-zero result

   Example:

   MVI A 10 (load 10H in register A)
   SUB A (A = A – A)
   These set of instructions will set the zero flag to 1 as 10H – 10H is 00H

3. Auxiliary Carry Flag (AC) – This flag is used in BCD number system(0-9). If after any arithmetic or logical operation D(3) generates any carry and passes on to B(4) this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. This is the only flag register which is not accessible by the programmer
   1-carry out from bit 3 on addition or borrow into bit 3 on subtraction
   0-otherwise

   Example:

   MOV A 2B (load 2BH in register A)
   MOV B 39 (load 39H in register B)
   ADD B (A = A + B)
   These set of instructions will set the auxiliary carry flag to 1, as on adding 2B and 39, addition of lower order nibbles B and 9 will generate a carry.

4. Parity Flag (P) – If after any arithmetic or logical operation the result has even parity, an even number of 1 bits, the parity register becomes set i.e. 1, otherwise it becomes reset i.e. 0.
   1-accumulator has even number of 1 bits
   0-accumulator has odd parity

   Example:

   MVI A 05 (load 05H in register A)
   This instruction will set the parity flag to 1 as the BCD code of 05H is 00000101, which contains even number of ones i.e. 2.

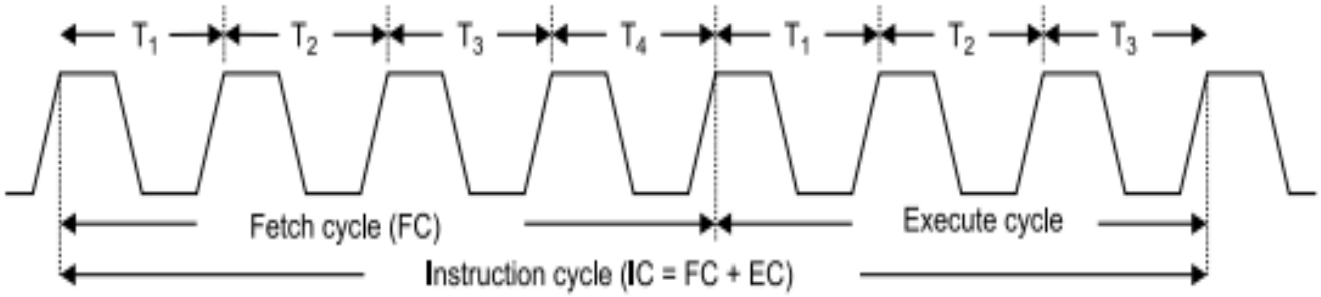
5. **Carry Flag (CY) –** Carry is generated when performing n bit operations and the result is more than n bits, then this flag becomes set i.e. 1, otherwise it becomes reset i.e. 0. During subtraction (A-B), if A>B it becomes reset and if (A<B) it becomes set. Carry flag is also called borrow flag.
   1-carry out from MSB bit on addition or borrow into MSB bit on subtraction
   0-no carry out or borrow into MSB bit

   Example:

   MVI A 30 (load 30H in register A)
   MVI B 40 (load 40H in register B)
   SUB B (A = A – B)
   These set of instructions will set the carry flag to 1 as 30 – 40 generates a carry/borrow.

   MVI A 40 (load 40H in register A)
   MVI B 30 (load 30H in register B)
   SUB B (A = A – B)
   These set of instructions will reset the sign flag to 0 as 40 – 30 does not generate any carry/borrow.

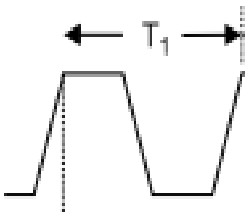
Timing Diagrams of 8085



- ✓ It is one of the best ways to understand to process of micro-processor/controller. With the help of timing diagram, we can understand the working of any system, step by step working of each instruction and its execution, etc.
- ✓ It is the graphical representation of process in steps with respect to time. The timing diagram represents the clock cycle and duration, delay, content of address bus and data bus, type of operation ie. Read/write/status signals.
- ✓ Important terms related to timing diagrams:
  - o **Instruction cycle:** this term is defined as the number of steps required by the cpu to complete the entire process ie. Fetching and execution of one instruction. The fetch and execute cycles are carried out in synchronization with the clock.

Instruction cycle= Fetch Cycle(FC) + Execute cycle(EC)

  - o **Machine cycle:** It is the time required by the microprocessor to complete the operation of accessing the memory devices or I/O devices. In machine cycle various operations like opcode fetch, memory read, memory write, I/O read, I/O write are performed.



Instruction = Opcode fetch + Operand

Operand = memory read, memory write, I/O read, I/O write

  - o **T-state:** Each clock cycle is called as T-states.

- ✓ **Rules to identify number of machine cycles in an instruction:**
  - o If an addressing mode is direct, immediate or implicit then

No. of machine cycles = No. of bytes.

- If the addressing mode is indirect then

  No. of machine cycles = No. of bytes + 1.

  Add **+1 to the No. of machine** cycles if it **is memory read/write** operation

- If the operand is 8-bit or 16-bit address then,

  No. of machine cycles = No. of bytes +1.

- These rules are applicable to 80% of the instructions of 8085.

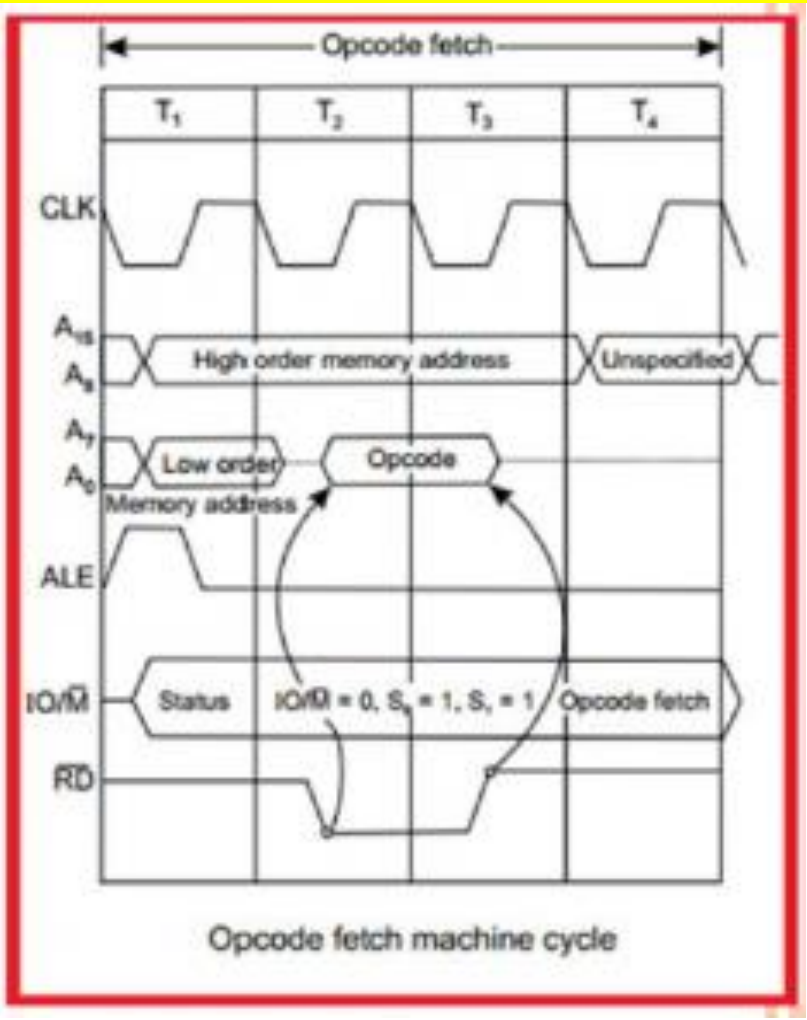
✓ In bellow table shows the **status of different control signal** for different operation. We should remember that to complete our timing diagram of 8085 microprocessors.

	Status			*Controls*		
Machine cycle	IO / $\overline{M}$	S_1	S_0	$\overline{RD}$	$\overline{WR}$	$\overline{INTA}$
Opcode Fetch (OF)	0	1	1	0	1	1
Memory Read	0	1	0	0	1	1
Memory Write	0	0	1	1	0	1
I/O Read (I/OR)	1	1	0	0	1	1
I/O Write (I/OW)	1	0	1	1	0	1

Opcode Fetch

- The lower byte of address (AD0 – AD7) is available on the multiplexed address/data bus during T1 state of each machine cycle, except during the bus idle machine cycle.
- The higher byte of address (A8 – A15) is available during T1 to T3 states of each machine cycle, except during the bus idle machine cycle, shown in Fig

- The first machine cycle of every instruction is the Opcode Fetch. This indicates the kind of instruction to be executed by the system.
- The length of this machine cycle varies between 4T to 6T states—it depends on the type of instruction. In this, the processor places the contents of the PC on the address lines, identifies the nature of machine cycle æ (by IO/M, S0, S1) and activates the ALE signal.
- All these occur in T1 State In T2 state, RD signal is activated so that the identified memory location is read from and places the content on the data bus (D0 – D7).
- In T3, data on the data bus is put into the instruction register (IR) and also raises the RD^ signal thereby disabling the memory.
- In T4, the processor takes the decision, on the basis of decoding the IR, whether to enter into T5 and T6 or to enter T1 of the next machine cycle.
- One byte instructions that operate on eight bit data are executed in T4. Examples are ADD B, MOV C, B, RRC, DCR C,etc.



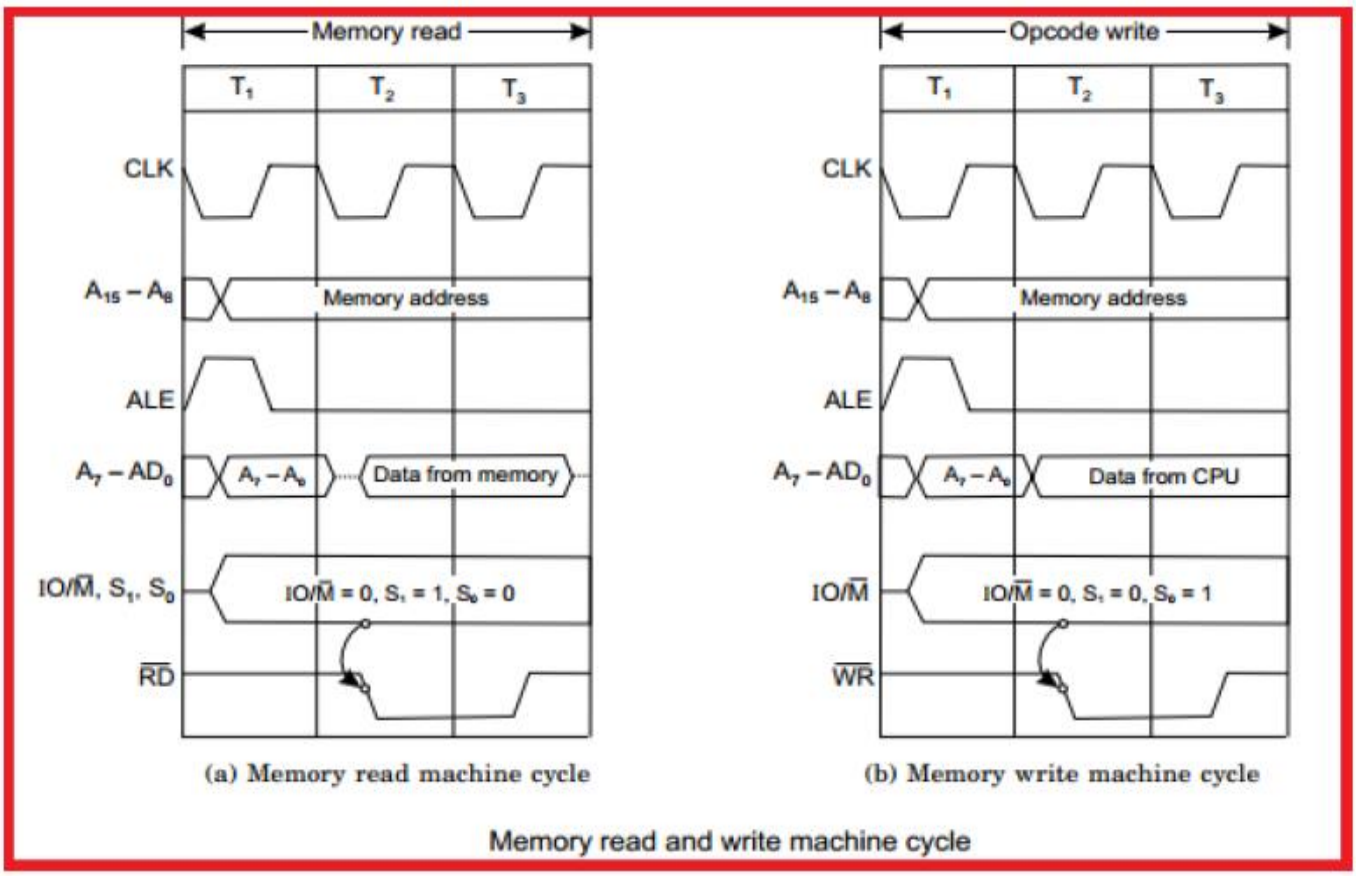
Opcode fetch machine cycle

Execute Cycle

Now see an example of memory read and memory write machine cycle.

- Both the Memory Read and Memory Write machine cycles are 3T states in length.
- In Memory Read the contents of R/W memory (including stack also) or ROM are read while in Memory Write, it stores data into data memory.
- As is evident from Fig during T2 and T3 states data from either memory or CPU are made available in Memory Read or Memory Write machine cycles respectively.

- The status signal (IO/ M, S0, S1) states are complementary in nature in Memory Read and Memory Write cycles. Reading or writing operations are performed in T2.
- In T3 of Memory Read, data from data bus are placed into the specified register (A,B, C, etc.) and raises RD so that memory is disabled while in T3 of Memory Write WR^ signal is raised which disables the memory.



Alternatively:

Opcode fetch:

 The microprocessor requires instructions to perform any particular action. In order to perform these actions microprocessor utilizes Opcode which is a part of an instruction which provides detail(ie. Which operation µp needs to perform) to microprocessor.

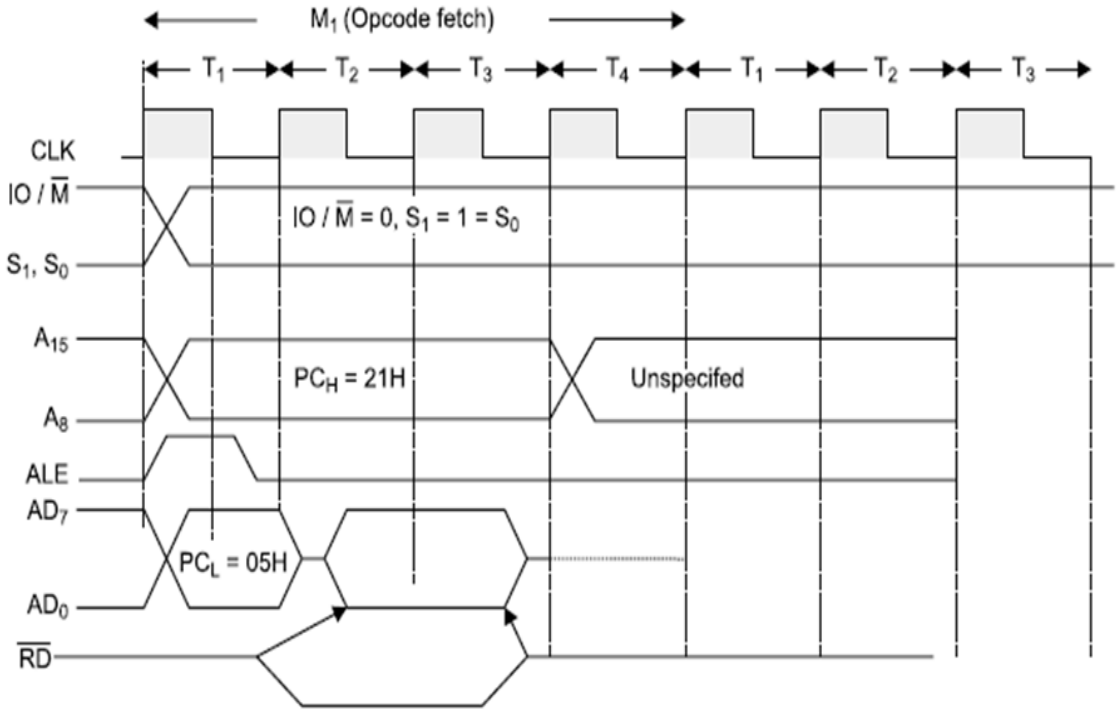


Fig: Opcode fetch timing diagram

Operation:

 During T1 state, microprocessor uses IO/M(bar), S0, S1 signals are used to instruct microprocessor to fetch opcode.

 Thus when IO/M(bar)=0, S0=S1= 1, it indicates opcode fetch operation.

 During this operation 8085 transmits 16-bit address and also uses ALE signal for address latching.

 At T2 state microprocessor uses read signal and make data ready from that memory location to read opcode from memory and at the same time program counter increments by 1 and points next instruction to be fetched.

 In this state microprocessor also checks READY input signal, if this pin is at low logic level ie. '0' then microprocessor adds wait state immediately between T2 and T3.

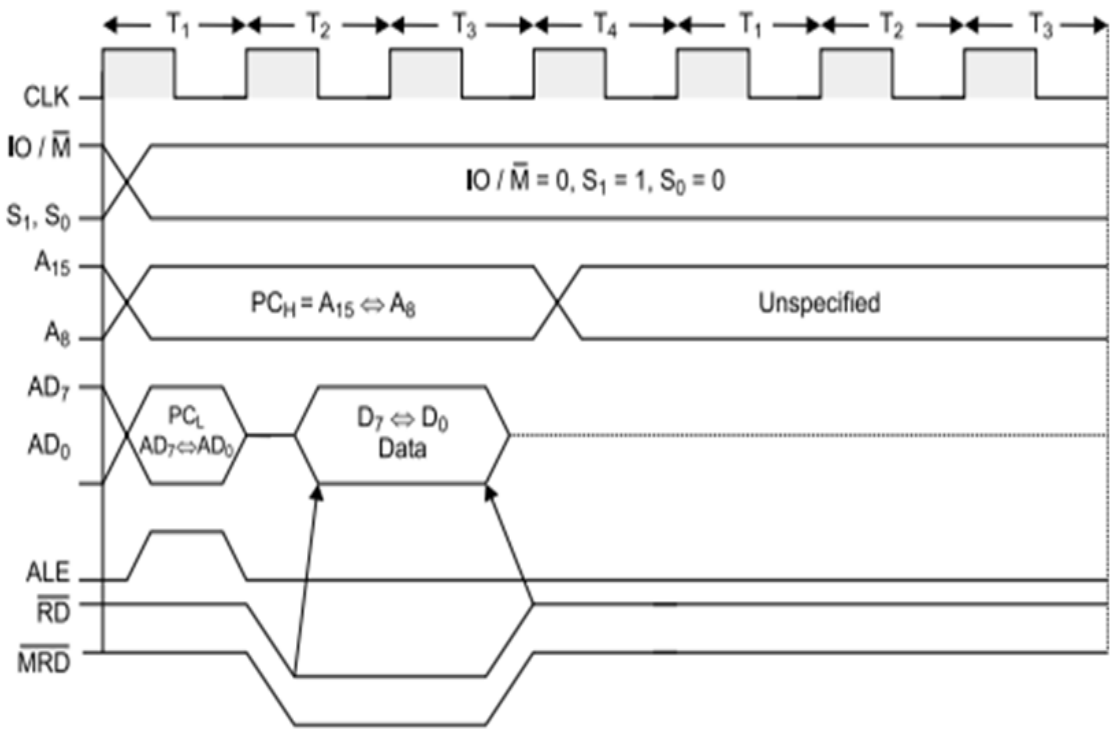
 At T3, microprocessor reads opcode and store it into instruction register to decode it further.

 During T4 microprocessor performs internal operation like decoding opcode and providing necessary actions.

 The opcode is decoded to know whether T5 or T6 states are required, if they are not required then µp performs next operation.

Read and write timing diagram for memory and I/O Operation

Memory



Read: <>Figure : Memory read timing diagram

Operation:

 It is used to fetch one byte from the memory.

- It requires 3 T-States.

- It can be used to fetch operand or data from the memory.

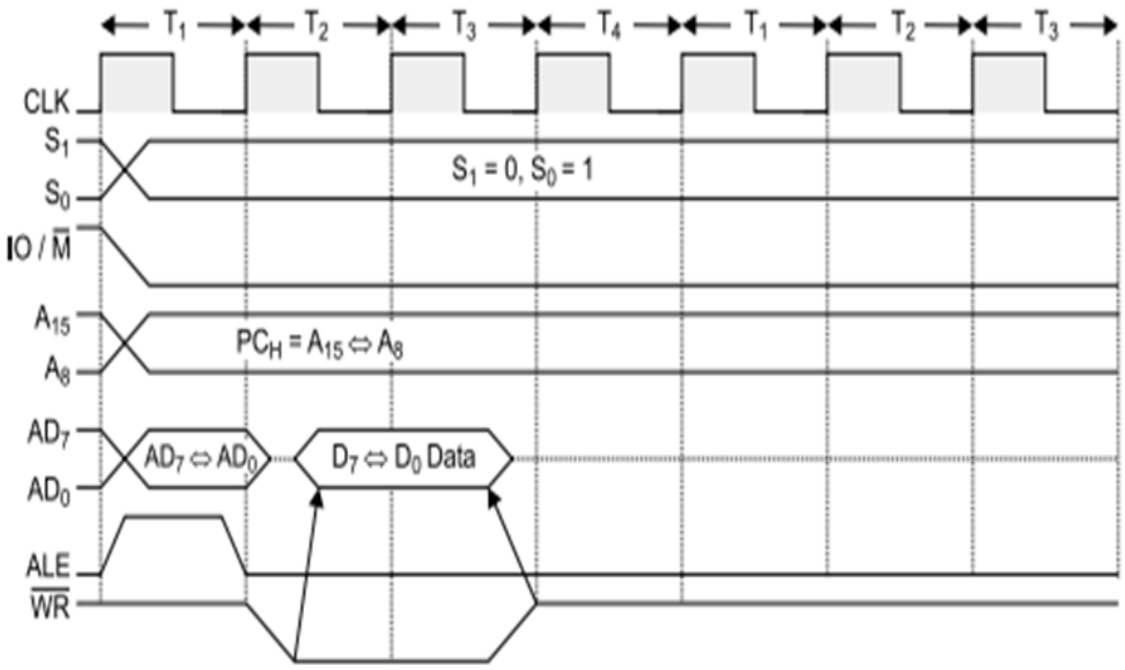
- During T1, A8-A15 contains higher byte of address. At the same time ALE is high. Therefore Lower byte of address A0-A7 is selected from AD0-AD7.

- Since it is memory ready operation, IO/M(bar) goes low.

- During T2 ALE goes low, RD(bar) goes low. Address is removed from AD0-AD7 and data D0-D7 appears on AD0-AD7.

- During T3, Data remains on AD0-AD7 till RD(bar) is at low signal.

Memory



Write:

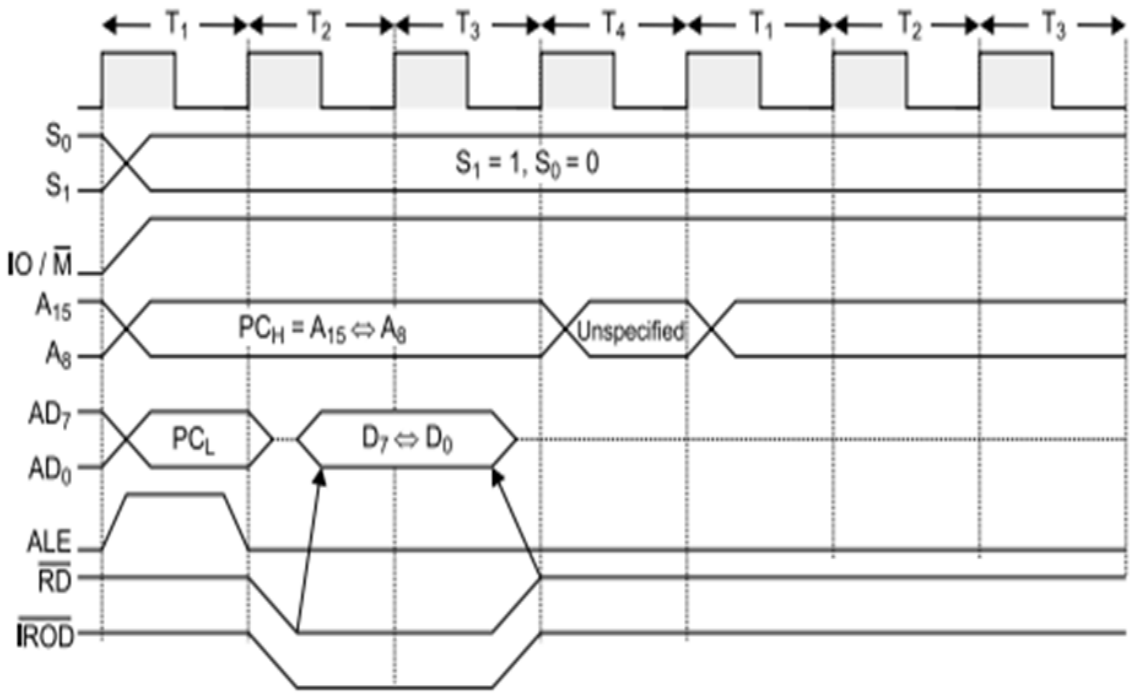
Figure: Memory write timing diagram

Operation:

- It is used to send one byte into memory.

- It requires 3 T-States.
- During T1, ALE is high and contains lower address A0-A7 from AD0-AD7.
- A8-A15 contains higher byte of address.
- As it is memory operation, IO/M(bar) goes low.
- During T2, ALE goes low, WR(bar) goes low and Address is removed from AD0-AD7 and then data appears on AD0-AD7.
- Data remains on AD0-AD7 till WR(bar) is low.

IO



Read: *Figure: I/O read timing diagram*

Operation:

It is used to fetch one byte from an IO port.

It requires 3 T-States.

During T1, The Lower Byte of IO address is duplicated into higher order address bus A8-A15.

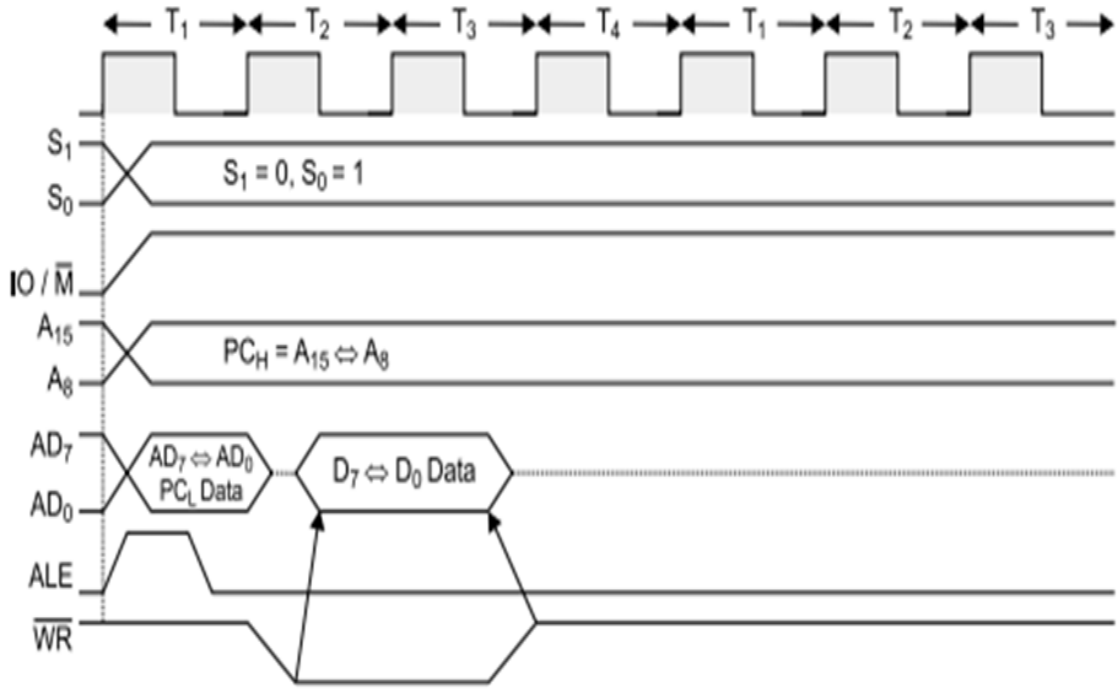
ALE is high and AD0-AD7 contains address of IO device.

IO/M (bar) goes high as it is an IO operation.

During T2, ALE goes low, RD (bar) goes low and data appears on AD0-AD7 as input from IO device.

During T3 Data remains on AD0-AD7 till RD(bar) is low.

IO



Write:

Figure:I/O write timing diagram

Operation:

- It is used to writ one byte into IO device.
- It requires 3 T-States.
- During T1, the lower byte of address is duplicated into higher order address bus A8-A15.
- ALE is high and A0-A7 address is selected from AD0-AD7.
- As it is an IO operation IO/M (bar) goes low.
- During T2, ALE goes low, WR (bar) goes low and data appears on AD0-AD7 to write data into IO device.
- During T3, Data remains on AD0-AD7 till WR(bar) is low.

Table: Important Intel Microprocessors

Microprocessor	Year of Invention	Word Length	Memory addressing Capacity	Pins	Clock	Remarks
4004	1971	4-bit	1 KB	16	750 KHz	First M
8085	1976	8-bit	64 KB	40	3-6 MHz	Popula Microp
8086	1978	16-bit	1MB	40	5-8 MHz	Widely
80286	1982	16-bit	16MB real, 4 GB virtual	68	6-12.5 MHz	Widely
80386	1985	32-bit	4GB real, 64TB virtual	132 14X14 PGA	20-33 MHz	Contai
80486	1989	32-bit	4GB real, 64TB virtual	168 17X17 PGA	25-100 MHz	Contai FPU, 1 transis
Pentium	1993	32-bit	4GB real,32-bit address,64-bit data bus	237 PGA	60-200	Contai Caches transis million
Pentium Pro	1995	32-bit	64GB real, 36-bit address bus	387 PGA	150-200 MHz	It is a proces second also,3.
Pentium II	1997	32-bit	-	-	233-400 MHz	All feat plus M V, 7.5

Pentium III	1999	32-bit	64GB	370 PGA	600-1.3 MHz	Improv Pentiu instruc
Pentium 4	2000	32-bit	64GB	423 PGA	600-1.3 GHz	Improv Pentiu
Itanium	2001	64-bit	64 address lines	423 PGA	733 MHz-1.3 GHz	64-bit

Where,

- **PGA** - Pin Grid Array
- **MMX** - MultiMedia eXtensions
- **EPIC** - Explicitly Parallel Instruction Computing
- **SIMD** - Single Instruction Multiple Data
- **ALU** - Arithmetic and Logic Unit
- **MMU** - Memory Management Unit
- **FPU** - Floating Point Unit

Basic Terms used in Microprocessor

Here is a list of some basic terms used in microprocessor:

Instruction Set - The group of commands that the microprocessor can understand is called Instruction set. It is an interface between hardware and software.

Bus - Set of conductors intended to transmit data, address or control information to different elements in a microprocessor. A microprocessor will have three types of buses, i.e., data bus, address bus, and control bus.

IPC (Instructions Per Cycle) - It is a measure of how many instructions a CPU is capable of executing in a single clock.

Clock Speed - It is the number of operations per second the processor can perform. It can be expressed in megahertz (MHz) or gigahertz (GHz). It is also called the Clock Rate.

Bandwidth - The number of bits processed in a single instruction is called Bandwidth.

Word Length - The number of bits the processor can process at a time is called the word length of the processor. 8-bit Microprocessor may process 8 -bit data at a time. The range of word length is from 4 bits to 64 bits depending upon the type of the microcomputer.

Data Types - The microprocessor supports multiple data type formats like binary, ASCII, signed and unsigned numbers.

Working of Microprocessor

The microprocessor follows a sequence to execute the instruction: Fetch, Decode, and then Execute.

Initially, the instructions are stored in the storage memory of the computer in sequential order. The microprocessor fetches those instructions from the stored area (memory), then decodes it and executes those instructions till STOP instruction is met. Then, it sends the result in binary form to the output port. Between these processes, the register stores the temporary data and ALU (Arithmetic and Logic Unit) performs the computing functions.

Features of Microprocessor

- **Low Cost** - Due to integrated circuit technology microprocessors are available at very low cost. It will reduce the cost of a computer system.
- **High Speed** - Due to the technology involved in it, the microprocessor can work at very high speed. It can execute millions of instructions per second.
- **Small Size** - A microprocessor is fabricated in a very less footprint due to very large scale and ultra large scale integration technology. Because of this, the size of the computer system is reduced.
- **Versatile** - The same chip can be used for several applications, therefore, microprocessors are versatile.
- **Low Power Consumption** - Microprocessors are using metal oxide semiconductor technology, which consumes less power.
- **Less Heat Generation** - Microprocessors uses semiconductor technology which will not emit much heat as compared to vacuum tube devices.
- **Reliable** - Since microprocessors use semiconductor technology, therefore, the failure rate is very less. Hence it is very reliable.
- **Portable** - Due to the small size and low power consumption microprocessors are portable.

Types of Microprocessors

Vector Processors

A **vector processor** is designed for vector computations. A vector is an array of operands of the same type. Consider the following vectors:

$$\text{Vector A } (a_1, a_2, a_3, \ldots\ldots, a_n)$$
$$\text{Vector B } (b_1, b_2, b_3, \ldots\ldots, b_n)$$
$$\text{Vector C = Vector A + Vector B}$$
$$= C(c_1, c_2, c_3, \ldots\ldots, c_n), \text{ where } c_1 = a_1 + b_1,\ c_2 = a_2 + b_2,\ \ldots\ldots, C_n = a_n + b_n.$$

A vector processor adds all the elements of vector A and Vector B using a single vector instruction with **hardware approach**.

Examples of vector processors are:

- DEC's VAX 9000,
- IBM 390/VF,
- CRAY Research Y-MP family,
- Hitachi's S-810/20, etc.

Array Processors or SIMD Processors

Array processors are also designed for vector computations. The difference between an array processor and a vector processor is that a vector processor uses multiple vector pipelines whereas an array processor employs a number of processing elements to operate in parallel.

An array processor contains multiple numbers of ALUs. Each ALU is provided with the local memory. The ALU together with the local memory is called a Processing Element (PE). An array processor is a **SIMD (Single Instruction Multiple Data)** processor. Thus using a single instruction, the same operation can be performed on an array of data which makes it suitable for vector computations.

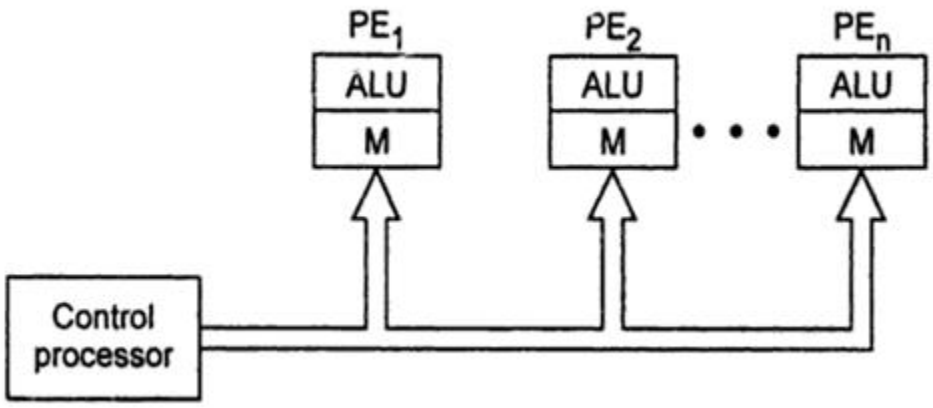


Fig:- Schematic Diagram of an Array Processor or SIMD Processor

---

Scalar and Superscalar Processors

A processor that executes scalar data is called scalar processor. The simplest scalar processor makes processing of only integer instruction using fixed-points operands. A powerful scalar processor makes processing of both integer as well floating- point numbers. It contains an integer ALU and a **Floating Point Unit (FPU)** on the same CPU chip.

A scalar processor may be RISC processor or CISC processor.

Examples of CISC processors are:

- Intel 386, 486; Motorola's 68030, 68040; etc.

Examples of RISC scalar processors are:

- Intel i860, Motorola MC8810, SUN's SPARC CY7C601, etc.

A **superscalar processor** has multiple pipelines and executes more than one instruction per clock cycle.

Examples of superscalar processors are:

- Pentium, Pentium Pro, Pentium II, Pentium III, etc.

RISC and CISC Processors

RISC stands for **Reduced Instruction Set Computer and**

CISC stands for **Complex Instruction Set Computer**.

There are two approaches of the design of the control unit of a microprocessor i.e.-

- Hardware approach and
- Software approach.

RISC Processors:- To execute an instruction, a number of steps are required. By the control unit of the processor, a number of control signals are generated for each step. To execute each instruction, if there is a separate electronic circuitry in the control unit, which produces all the necessary signals, this approach of the design of the control section of the processor is called **RISC** design. It is hardware approach. It is also called hard-wired approach.

Examples of RISC processors are:

- DEC's Alpha 21064, 21164 and 21264 processors;
- SUN's SPARC and ULTRA SPARC;
- PowerPC processors etc.

CISC Processors:- If the control unit contains a number of micro electronic circuitry to generate a set of control signals and each micro circuitry is activated by a microcode, this design approach is called CISC design. This is a software approach of designing a control unit of the processor.

Examples of CISC processors are:

- Intel 386, 486;
- Pentium Pro, Pentium, Pentium II, Pentium III, Pentium 4;
- Motorola's 68000, 68020, 68030, 68040, etc.

Difference between RISC and CISC

S.No.	RISC	CISC
1.	Simple instruction set	Complex instruction set
2.	Consists of Large number of registers.	Less number of registers
3.	Larger Program	Smaller program
4.	Simple processor circuitry (small number of transistors)	Complex processor circuitry (more transistors)
5.	More RAM usage	Little Ram usage
6.	Simple addressing modes	Variety of addressing modes
7.	Fixed length instructions	Variable length instructions
8.	Fixed number of clock cycles for executing one instruction	Variable number of clock cycles for

Digital Signal Processors (DSP)

DSP microprocessors specifically designed to process signals. They receive some digitized signal information, perform some mathematical operations on the information and give the result to an output device. They implement integration, differentiation, complex fast Fourier transform, etc. using hardware.

Examples of digital signal processors are:

- Texas instruments' TMS 320C25,
- Motorola 56000,
- National LM 32900,
- Fujitsu MBB 8764, etc.

Symbolic Processors

Symbolic processors are designed for expert system, machine intelligence, knowledge based system, pattern-recognition, text retrieval, etc.

The basic operations which are performed for artificial intelligence are:

Logic interference, compare, search, pattern matching, filtering, unification, retrieval, reasoning, etc. This type of processing does not require floating point operations. Symbolic processors are also called **LISP processors or PROLOG processors**.

Bit-Slice Processors

The processor of desired word length is developed using the building blocks. The basic building block is called Bit-Slice where the building blocks include 4-bit ALUs, micro programs sequencers, carry look-ahead generators, etc. The word 'slice' was used because the desired number of ALUs and other components were used to build an 8-bit, 16-bit or 32-bit CPU.

Examples of Bit-Slice Processors were:

- AMD-2900, AMD 2909, AMD 2910, AMD 29300 series,
- Texas instrument's SN-74AS88XX series, etc.

Transputers

In a multiprocessor system, a transputer is a specially designed microprocessor to operate as a component processor.

Transputers were introduced in late 1980's. They were built on VLSI chip and contained a processor, memory and communication links. The communication link was to provide point-to-point connection between transputers.

A transputer contains FPU, on-chip RAM, high-speed serial link, etc.

Examples of transputers are:

- INMOS T414, INMOS T800, etc.

Where, T414 was a 32-bit processor with 2 KB memory. The T800 was FPU version of 32-bit transputer with 4 KB memory.

Graphic Processors

Graphics Processors are specially designed processors for graphics. Intel has developed Intel 740-3D graphics chip. It is optimized for Pentium II PCs, using a hyper pipelined 3D architecture with additional

2D acceleration. Like most 3D graphics chips, the I-740 will be marketed in performance, not the main stream category. It is designed mostly for such heavy multimedia uses as games and movies.

Examples of Graphic Processors are:

- Intel 82786 graphics coprocessor
- IBM's 8514/A,
- Texas Instruments' TMS34010 and TMS34020,
- Intel i860 and Intel i750, etc.