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**

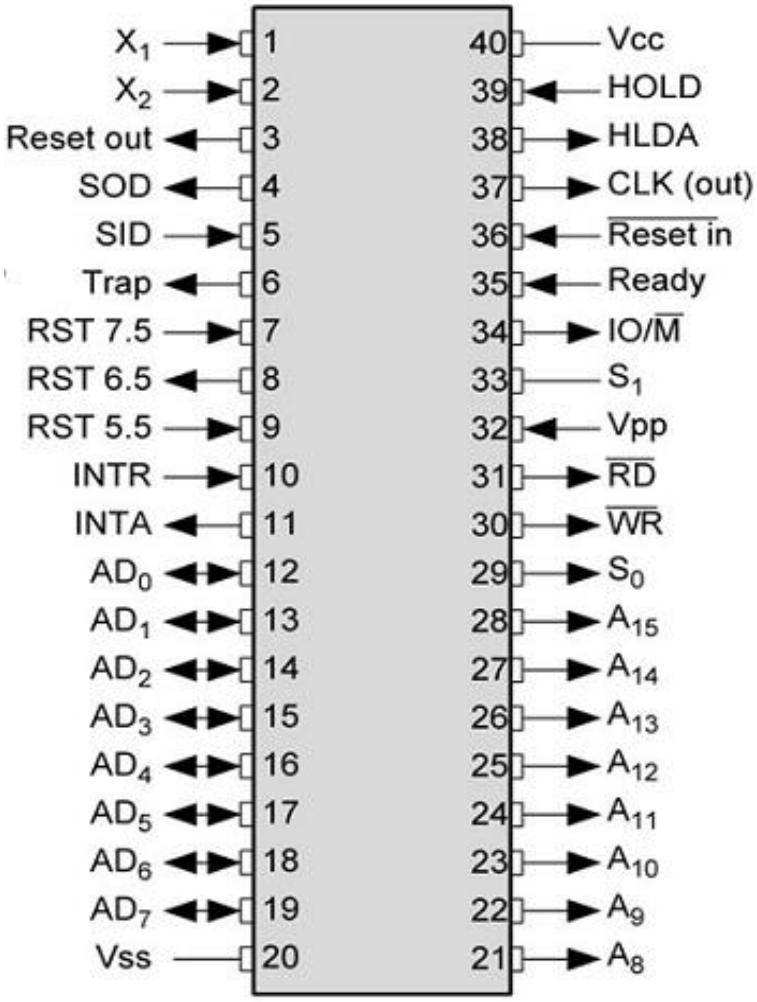
   A15-A8, it carries the most significant 8-bits of memory/IO address.

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.

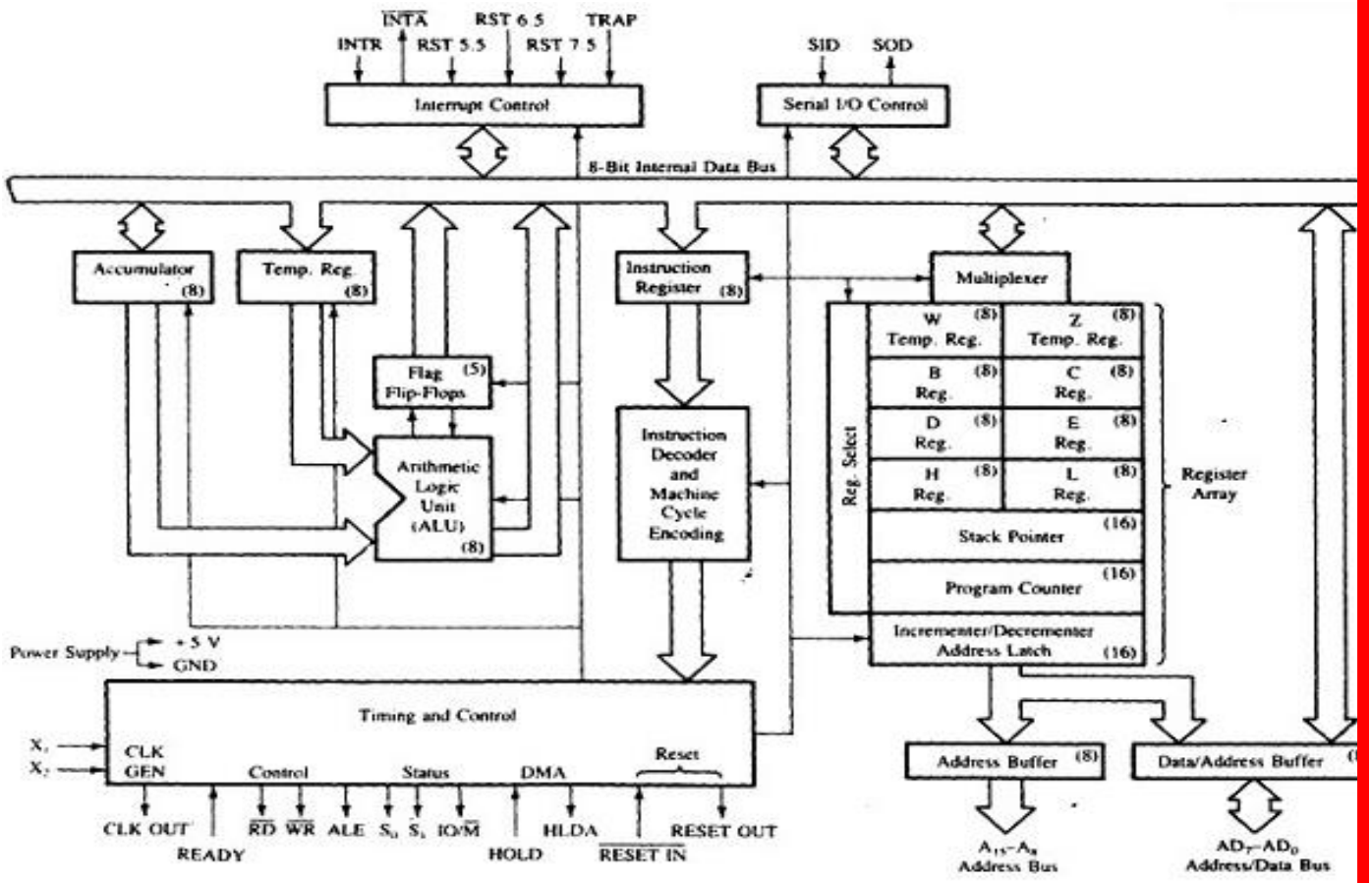
Group 4:

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

These are the instructions used to transfer the data from one register to another register, from the memory to the register, and from the register to the memory without any alteration in the content. Addressing modes in 8085 is classified into 5 groups –

Immediate addressing mode

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operand. **For example:** MVI K, 20F: means 20F is copied into register K.

Register addressing mode

In this mode, the data is copied from one register to another. **For example:** MOV K, B: means data in register B is copied to register K.

Direct addressing mode

In this mode, the data is directly copied from the given address to the register. **For example:** LDB 5000K: means the data at address 5000K is copied to register B.

Indirect addressing mode

In this mode, the data is transferred from one register to another by using the address pointed by the register. **For example:** MOV K, B: means data is transferred from the memory address pointed by the register to the register K.

Implied addressing mode

This mode doesn't require any operand; the data is specified by the opcode itself. **For example:** CMP.

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.

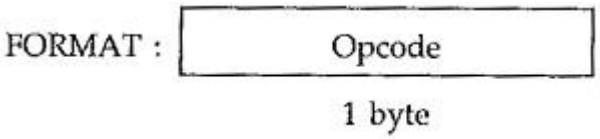


Table 2.1 Example for 1 byte Instruction

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.

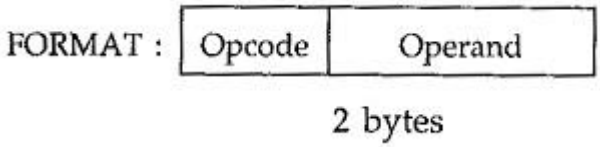


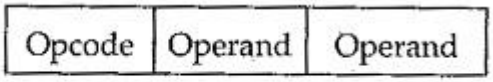
Table 2.2 Example for 2 byte Instruction

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 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

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**	Subtract source and	The contents of the operand (register or memory) and M the Borrow flag are subtracted from the contents of the

	M	borrow from accumulator	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 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 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

Opcode	Operand	Explanation of Instruction	Description

<table>
<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 program. The two bytes from the top of the stack are copied into the program counter,and</td></tr>
</table>

			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"
```