Chapter 1

BASICS

A) Logic Gates (AND, OR, NOT, NAND, NOR, EX-OR): Review of all logic gates; AND, OR, NOT, NAND, NOR, EX-OR & their truth tables. Appropriate combinations of gates results into an amazing & innovative logical configuration.

B) Number Systems (Binary, Octal, Decimal & Hexadecimal): In digital, we normally deal with four number systems of arithmetic (I) Binary (II) Octal (III) Decimal (IV) Hexadecimal. The commonly used number system by all of us is decimal, while the binary number system is used by computers.

Number Systems & Equivalence

Decimal (10)	Octal (8)	Hexadecimal (16)	Binary (2)
0	0	0	0000 0000
1	1	1	0000 0001
2	2	2	0000 0010
3	3	3	0000 0011
4	4	4	0000 0100
5	5	5	0000 0101
6	6	6	0000 0110
7	7	7	0000 0111
8	10	8	0000 1000
9	11	9	0000 1001
10	12	A	0000 1010
11	13	B	0000 1011
12	14	C	0000 1100
13	15	D	0000 1101
14	16	E	0000 1110
15	17	F	0000 1111

Refer the table above, it is obvious that:

- $(10)_{10}=(12)_8=(A)_{16}=(1010)_2$; $(15)_{10}=(17)_8=(F)_{16}=(1111)_2$; And so on…….

C) Bit, a Nibble, a Byte, 1's Complement & 2's Complement:

Bit: The smallest unit of data in a computer is called Bit. A bit has a single binary value, either 0 or 1.

Nibble: Half a byte that is four bits is called a nibble.

Byte: Eight Bits forms a Byte.

1's Complement & 2's Complement; their Significance: Let's consider a number in Hexadecimal System: $(FB)_{16}$ = $(1111\ 1011)_2$. Further let's consider complement of its binary form i.e. $(0000\ 0100)_2$ let add 1 to this number i.e. $(0000\ 0101)_2$. Hence $(04)_{16}$ and $(05)_{16}$ are one's complement and two's complement of the number $(FB)_{16}$.

D) The alternative way is FF - FB = 04 is 1'scomplement and 04 + 01 = 05 is two's complement of $(FB)_{16}$. Further let's add FB & 05=100. If we neglect 1, the sum of a number and its 2'scomplement is "ZERO" but we know addition of a number to its negative only can lead to zero $\Longrightarrow$ 2's complement of a number is –Ve of the number. Computer does not subtract, it always adds, hence subtraction of number exactly means, addition of its 2s' complement.

E) Sign-Magnitude Representation: Refer the numbers 11100011 and 01100011. If the MSB is assigned for sign then, $(1\ 1100011) = (-63)_{16}$ and $(0\ 1100011) = (+63)_{16}$. In a decimal number system, a (+) sign is used to denote positive number and (-) sign is used to denote negative number. In digital system, the sign of the binary number is also represented by 0 & 1. In signed binary number system, the numbers also have the sign, MSB '0' is to represent positive number and '1' is to represent negative number.

F) Parity its Significance: Let's consider a number in hexadecimal system $(F2)_{16}$ = $(1111\ 0010)_2$. The number of 1s in its binary equivalent is "5", hence, the number F2 is with odd parity. Similarly let's consider another number $(B5)_{16}$ = $(1011\ 0101)_2$ is again with an odd parity. While $(C3)_{16}$ = $(1100\ 0011)_2$ is with an even parity. $\Longrightarrow$ Number of 1s in the binary equivalent of a hexadecimal number decides the parity of the number.

G) LSB and MSB of a Binary Number & its Significance: The LSB and MSB of binary number has its own significance altogether. Refer the numbers $(0010)_2$, $(0100)_2$, $(0110)_2$, $(1000)_2$ …… the even numbers are those whose LSB is '0'. Refer the numbers $(0011)_2$, $(0101)_2$, $(0111)_2$ ……, the odd numbers are those whose LSB is one. Similarly, if MSB is '1' the number is negative and when it is '0' it is positive.

H) Binary Digits: The microprocessor operates on binary digits, 0 and 1, also known as bits. Bit is an abbreviation for the term binary digit. These digits are represented in terms of electrical voltages in machine: Generally, 0 represents low voltage level & 1 represents high voltage level. The digits 0 and 1 are also synonymous with low & high respectively.

Chapter 2

μP 8085: HARDWARE & SOFTWARE

The microprocessor is a device consisting of electronic logic circuits manufactured by using either a (LSI) large-scale integration or (VLSI) very-large-scale-integration technique. The microprocessor can be divided into three segments for the sake of clarity; arithmetic/logic unit (ALU), register array & control unit.

A multipurpose, programmable, clock-driven, register-based device that reads binary instruction from a storage device called memory, accepts binary data as input and processes data according to those instructions and provides results as output.

At very elementary level, we can draw an analogy between microprocessor operations and the functions of a human brain that process information according to instructions (understanding) stored in its memory. The brain gets input from eyes and ears and sends processed information to output "devices" such as the face, hands or feet. A typical programmable machine can be represented with four components: Microprocessor, Memory, Input and Output. These four components work together or interact with each other to perform a given task.

The physical components of the system are called hardware. A set of instruction written for the microprocessor to perform a task is called program and a group of programs is called software. The microprocessor applications are classified primarily into two categories: Reprogrammable Systems & Embedded Systems.

In reprogrammable systems, such as microcomputers, the microprocessor is used for computing and data processing. These systems include general-purpose microprocessors capable of holding large data, mass storage devices (such as disks and CD-ROMs), and peripherals such as printers; a personal computer (PC) is a typical illustration.

Embedded systems can also be viewed as products that use microprocessors to perform their operations; they are known as microprocessor-based products. Example includes a wide range of products such as Washing machines, Dishwashers, Automobile dashboard controls, Traffic light controllers & Automatic testing instruments etc.

- **Features of 8085, Pin diagram of 8085, CPU Structure, Registers of 8085:**

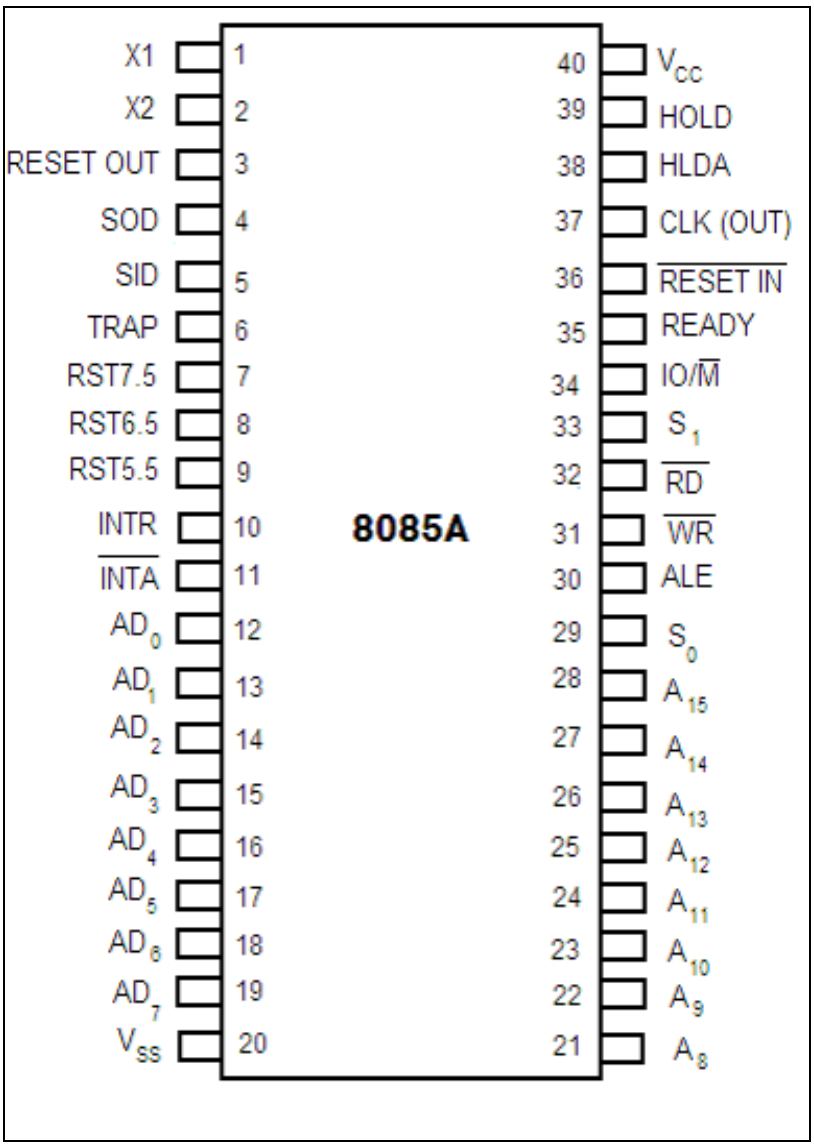
8085 & It's Buses:

- ✓ The 8085 is an 8-bit general purpose μP & can address 64 K Byte of memory.
- ✓ 4 maskable and 1 non-maskable interrupt.
- ✓ Direct Memory Addressing (DMA) capability.
- ✓ Can run at a maximum frequency of 3 MHz
- ✓ The pins on the chip can be grouped into six categories:
  - Address Bus.
  - Data Bus.
  - Control and Status Signals.
  - Power Supply and Frequency.
  - Externally Initiated Signals.
  - Serial I/O ports.

8085 Pin Description:

- Higher Order Address Pins: $A_{15} – A_8$
- The address bus has 8 signal lines $A_8 – A_{15}$ which are unidirectional.
- Lower Order Address / Data Pins- AD_7-AD_0
  - ✓ These are time multiplexed pins and are de-multiplexed using ALE
  - ✓ So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.

- ✓ During the execution of the instruction, during the early part these lines carry the address bits, then during the later parts of the execution they carry the 8-data bits.
- ✓ In order to separate the address from the data, we can use a latch to save the value before the function of the bit's changes.
- ✓ Control Pins: RD, WR; These are active low, Read & Write pins

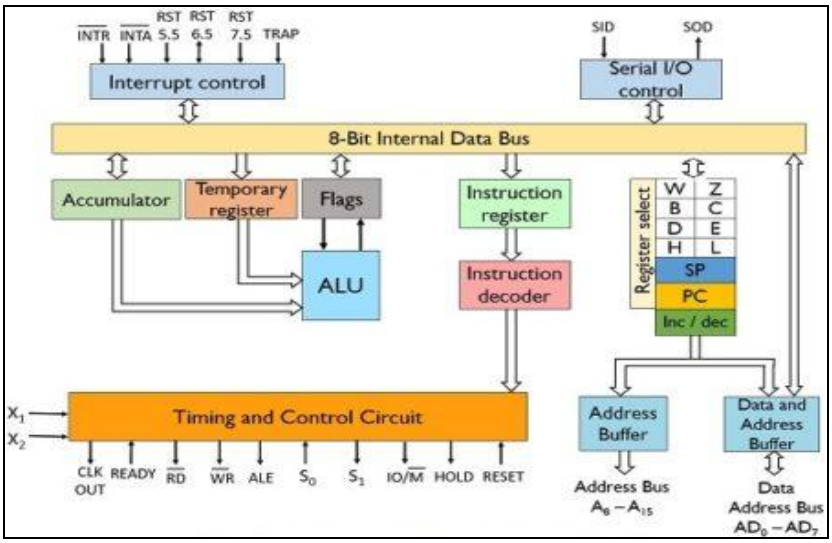


- Status Pins: ALE, IO/M (active low), S_1, S_0
  - ✓ ALE (Address Latch Enable)-Used to de-multiplex AD_7-AD_0
  - ✓ IO/M – Used to select I/O or Memory operation
  - ✓ S_1, S_0 – Denote the status of data on data bus
- Interrupt Pins: TRAP, RST7.5, RST 6.5, RST 5.5, INTR & INTA
  - ✓ These pins are for hardware interrupts used to initiate an interrupt service routine stored at predefined locations of the system memory.
- Serial I/O Pins: SID (Serial Input Data), SOD (Serial Output Data)
  - ✓ These pins are used to interface 8085 with a serial device.
- Clock Pins: X_1, X_2, CLK (OUT)
  - ✓ X_1, X_2 – These are clock input pins. A crystal is connected between these pins such that $f_{crystal} = 2f_{8085}$ where $f_{crystal}$ = crystal frequency & f_{8085} = operating frequency of 8085
  - ✓ CLK(OUT) – This is an auxiliary clock output source
- Reset Pins: Reset In (active low), Reset Out
  - ✓ Reset In is used to reset 8085 whereas Reset Out can be used to reset other devices in the system.
- DMA (Direct Memory Access) pins – HOLD, HLDA
  - ✓ These pins are used when data transfer is to be performed directly between an external device and the main memory of the system.
- Power Supply Pins: $+V_{CC}$, V_{SS}

8085 Configuration:

- ✓ 8-bit data bus & 16-bit address bus.
- ✓ 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.

Architecture of 8085:



- **Arithmetic & Logic Unit:** This is the area of the microprocessor where various computing functions are performed. The ALU performs arithmetic operations such as addition and subtraction and logic operations such as AND, OR and EX-OR.
- **Register Array:** This area of the microprocessor consists of various registers identified by letters such as A, B, C, D, E, H, L & F these registers are primarily used to store data temporarily during the execution of a program and accessible to the user through instructions, except F register.
- **Control Unit:** The control unit provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor, memory and peripherals.
- **Memory:** Memory stores such binary information as instructions and data and provides that information to the microprocessor whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Results are either transferred to the output section for display or stored in memory for later use.

The memory can be divided into two sections: Read-Only Memory (ROM) and Random-Access memory (RAM). The ROM is used to store programs that do not need alterations. The monitor program of a single-board microcomputer is generally stored in the ROM. This program interprets the information entered through a keyboard and provides equivalent binary digits to the microprocessor. Programs stored in the ROM can only be read; they cannot be altered. RAM is also known as user memory, used to store programs and data. In single board microcomputers, the monitor program monitors the hex key and stores those instructions and data in the RAM memory. The information stored in RAM can be easily read and altered.

I/O: The third component of a microprocessor-based system is I/O (input/output); It communicates with the outside world. I/O includes two types of devices: input and output; also known as peripherals. The input devices such as a keyboard, switches and an analog-to-digital (A/D) converter transfer binary information from the outside world to the microprocessor. Typically, a microcomputer used in college laboratories includes either a hexadecimal keyboard or an ASCII keyboard as an input device. The hexadecimal (Hex) keyboard has 16 data keys (0 to 9 and A to F) and some additional function keys to perform different operations such as storing data and executing programs.

The ASCII keyboard is like a typewriter keyboard and it is used to enter programs in an English-like language. Although the ASCII keyboards are found in most microcomputers (PCs). Single-board microcomputers generally have Hex keyboards and microprocessor-based products such as microwave oven have decimal keyboards.

The O/P devices transfer data from the microprocessor to the outside world. They include devices such as light emitting diodes (LEDs), a cathode-ray tube (CRT) or video screen, a printer, X-Y plotter, a magnetic tape, and digital-to-analog (D/A) converter. Single-board microcomputers and microprocessor-based products such as a dishwasher or microwave oven include

LEDs, seven-segment LEDs, and alphanumeric LED displays as output devices. Microcomputers (PCs) are generally equipped with output devices such as a video screen also called a monitor and a printer.

System Bus: The system bus is a communication path between the microprocessor and peripherals, it is nothing but the group of wires to carry bit. In fact, there are several buses in the system. All peripherals and memory share the same bus; however, the microprocessor communicates with only one peripheral at a time. The timing is provided by the control unit of the microprocessor.

Machine Language: The 8085 is a microprocessor with 8-bit word length; its instruction set is designed by using various combinations of these eight bits. The 8085 microprocessor has 256-bit patterns, amounting to 74 different instructions for performing various operations. These 74 different instructions form the instruction set.

Assembly Language: Each manufacturer of a microprocessor has devised a symbolic code for each instruction, called a mnemonic; the mnemonic for a particular instruction consists of letters that suggest the operation to be performed by that instruction.

Writing & Executing an ALP: A program is a set of logically related instructions written in a specific sequence to accomplish a task. To manually write and execute an assembly language program on a single-board computer, with a Hex keyboard for input and LEDs for output.

High-Level Languages: Programming languages that are intended to be machine independent are called high-level languages. These include such languages as BASIC, PASCAL, C, C++ & Java, all of which have certain sets of rules and draw on symbols from English. Instructions written in these languages are known as statements rather than mnemonics.

Microprocessor-Initiated Operations & 8085 Bus Organization:
The MPU performs primarily four operations:

- ✓ Memory Read: Reads data (or instructions) from memory.
- ✓ Memory Write: Writes data (or instructions) into memory.
- ✓ I/O Read: Accepts data from input devices.
- ✓ I/O Write: Sends data to output devices.

The 8085 MPU performs these functions using three sets of communication lines called buses: the address bus, data bus and control bus.

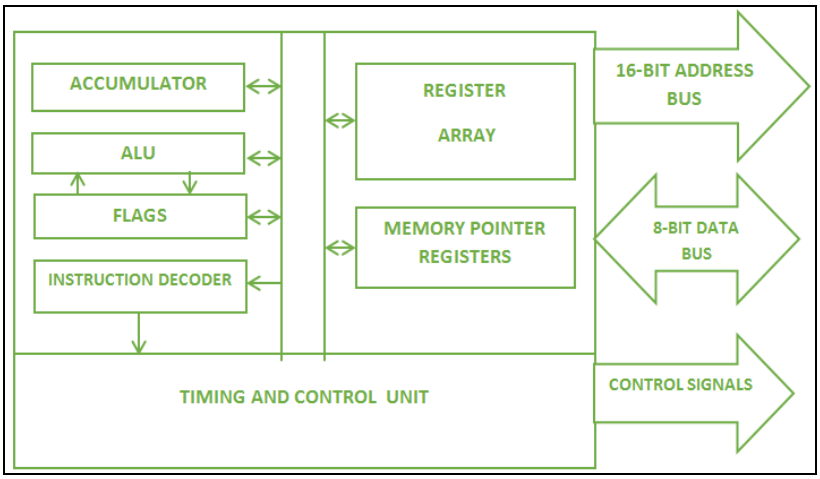
Address Bus: The address bus is a group of 16-lines generally identified as A_0 to A_{15}. The address bus is unidirectional; bits flow in one direction from MPU to peripheral devices. The MPU uses the address bus to perform the first function, identifying a peripheral or a memory location (Step 1).

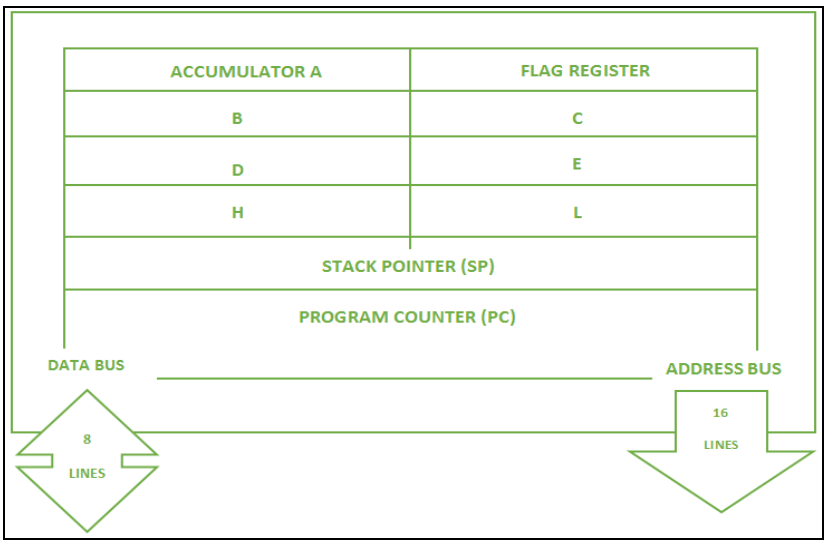
Data Bus: The data bus is a group of 8-lines used for data flow. These lines are bidirectional, data flow in both directions between the MPU and memory and peripheral devices. The MPU uses the data bus to perform the second function: transferring binary information (Step 2).

Control Bus: The control bus comprised of various single lines that carry synchronization signals. The MPU uses such lines to perform the third function: providing timing signals. The term bus, in relation to the control signals, is somewhat confusing. These are not groups of lines like address or data buses, but individual lines that provide a pulse to indicate an MPU operation. The MPU generates specific control signals for every operation it performs such as Memory Read or I/O Write. These signals are used to identify a device type with which the MPU intends to communicate.

8085 Hardware Model: The hardware model as shown below shows two major segments. One segment includes the arithmetic/logic unit (ALU) and an 8-bit register called as accumulator, instruction decoder and flags. The second segment shows 8-bit and 16-bit registers. Both segments relate to various internal connections called as internal bus.

Hardware & Programming models are shown below.





8085 Programming Model: The programming model consists of some segments of the ALU and the registers. This model does not reflect the physical structure of 8085 but includes the information that is critical in writing assembly language programs. The model includes six registers, one accumulator and one flag register, as shown in figure above. In addition, it has two 16-bit registers: the stack pointer and the program counter.

Registers: The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L . They can be combined as register pairs BC, DE and HL to perform some 16-bit

operations. The programmer can use these registers to store or copy data into registers by using data copy instructions.

Accumulator: The accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is also stored in the accumulator. The accumulator is also identified as register A.

Flags: The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. The most commonly used flags are Zero, Carry and Sign. The microprocessor uses these to test data conditions.

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

Flag Register

8085 Flags: The following flags are set or reset after the execution of an arithmetic or logic operation; data copy instructions do not affect any flags.

- Zero (Z): The Zero flag is set to 1 when the result is zero; otherwise it is reset.
- Carry (CY): If an arithmetic operation results in a carry, the CY flag is set; otherwise it is reset.
- Sign (S): The sign flag is set if bit D_7 of the result = 1; otherwise it is reset.
- Parity(P): If the result has an even number of 1s, the flag is set; for an odd number of 1s, the flag is reset.
- Auxiliary Carry (AC): In an arithmetic operation, when a carry is generated by digit D_3 and passed to digit D_4, the AC flag is set. This flag is used internally for BCD (binary-coded-decimal) operations; there is no Jump instruction associated with this flag.

Program Counter (PC) & Stack Pointer (SP): These are two 16-bit registers used to hold memory addresses. The size of these registers is 16-bit because the memory addresses are 16-bits. The microprocessor uses the PC register 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. The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in RAM memory called the stack. The beginning of the stack is defined by loading a 16-bit address in the stack pointer.

The 8085 MPU: The term micro-processing unit (MPU) is like the term central processing unit (CPU) used in traditional computers. We define the MPU as a device or a group of devices that can communicate with peripherals, provides timing signals, direct data flow and perform computing tasks as specified by the instructions in memory. The unit will have the necessary lines for the address bus, the data bus and the control signals and would require only a power supply and a crystal to be completely functional.

The ALU: The arithmetic/logic unit performs the computing functions: it includes the accumulator, the temporary register, the arithmetic and logic circuits and five flags. The temporary register is used to hold data during an arithmetic or logic operation. The result is stored in the accumulator and the flags are set or reset according to the result of the operation. The flags are affected by the arithmetic and logic operations in the ALU. In most of these operations, the result is stored in the accumulator. Therefore, the flags generally reflect data conditions in the accumulator with some exceptions.

Instruction: An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the instruction set, determines what functions the microprocessor can perform. Each instruction basically consists of two parts. The first part is called as op-code and the second is called as operand. The op-code part of an

instruction specifies the operation to be performed. The operand either provides the data or specifies data. The operand can be specified in many ways.

It includes:

- 8-bit/16-bit internal general-purpose registers.
- A memory location.
- 8-bit port address/ 16-bit memory address.
- Implicit operand: The operand is not specified, instead it is assumed in register.
- Instruction Word Size
- Opcode Format

The instruction consists of two parts opcode and operand as specified earlier. Depending the instructions are of 3 types:

1. One-byte instructions.
2. Two-byte instructions.
3. Three-byte instructions.

Addressing Modes: Immediate, Register, Direct & Indirect Addressing Modes

Addressing Modes	
Immediate	Data is present in the instruction. Load the immediate data to the destination provided. Example: MVI R, data
Register	Data is provided through the registers. Example: MOV R_d, R_s
Direct	Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or sends the data from the accumulator to the port 01H.
Indirect	This means that the effective address is calculated by the processor. And the contents of the address (and the one following) are used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions, the source can be register, an input port or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The source and destination are operands. The various formats for specifying operands are called the "Addressing Modes".

- **Immediate Addressing:** In Immediate addressing mode, the data to be used is immediately given in the instruction itself. The immediate addressing instructions are either 2- byte or 3-byte long. In 2-byte instruction first byte is opcode and second byte an 8-bit data. In 3-byte instruction first byte is opcode, second and third bytes are the 16-bit data.

Example:

- MVI A, 20H: This instruction transfers immediate data (20H) to a register.
- LXI H, C200H: This instruction transfers 16-bit immediate data C200H to HL register pair. Lower order data (00H) to L reg. & high order data (C2H) to H reg.

- **Register Addressing:** In register addressing mode the data to be operated is in the general-purpose registers. The register addressing instructions are generally of 1-byte i.e. opcode only. The opcode specifies the operation and registers to be used to perform the operation.

Example:

- MOV A, B: This instruction copies the content of register B to the A reg. i.e. accumulator. The data source and destination are both register.
- ADD B: This instruction adds the content of B reg. and A reg. The data is present in both B and A reg. In instruction only one register (B) is specified and other register is assumed to be present i.e. A reg.

- **Direct Addressing:** In direct addressing mode, the operand is given by a direct address where the data is present. The direct addressing mode instruction is 3-byte instruction. First

byte is an opcode, second is lower order address and third is higher order address.

Example:

- LDA C200H: Load accumulator directly from memory location. The address of memory location is specified in the instruction. In this instruction, the contents of C200H memory location are transferred to accumulator.
- STA C200H: Store accumulator directly to memory location. The address of memory location is specified in the instruction. In this instruction, the contents of accumulator are stored at C200H memory location.

➢ **Indirect Addressing:** In indirect addressing mode, the instruction does not have the address of the data to be operated on. But the instruction points where the address is stored i.e. it is indirectly specifying the address of memory location where the data is stored or is to be stored.

Example:

- MOV A, M: In this case M is a memory pointer specifying HL register pair where the address is stored. The contents of HL pair are used as address and the contents of that memory location are transferred to accumulator.
- LDAX B: In this case BC register pair is used as address and the contents of that memory location are transferred to accumulator.

The 8085 Instruction Set (Classification):

Notations	Meaning
M	Memory location pointed by HL register pair
R	8-bit register
R_p	16-bit register pair
R_s	Source register
R_d	Destination register
addr	16-bit address
Data	8-bit data

Notation @ opcode are:

Notation	Meaning
ddd	Destination Register
sss	Source Register
	Address of Registers
	111 = A reg.
	000 = B reg.
	001 = C reg.
	010 = D reg.
	011 = E reg.
	100 = H reg.
	101 = L reg.
nnn	Restart number 000 to 111
yy	An 8-bit binary data
yyyy	A 16-bit binary data unit
xx	Address of Register pair
	00 = BC
	01 = DE
	10 = HL
	11 = SP (if PUSH/POP) PSW
ppqq	A 16-bit memory address

An instruction is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions called the instruction set. The instructions can be classified into the following function categories.

1. Data Transfer OR Copy Operations.
2. Arithmetic Operations.
3. Logical Operations.
4. Branching Operations.

Chapter 3

UNDERSTANDINGS INSTRUCTIONS

A) Data Transfer / Copy Instructions

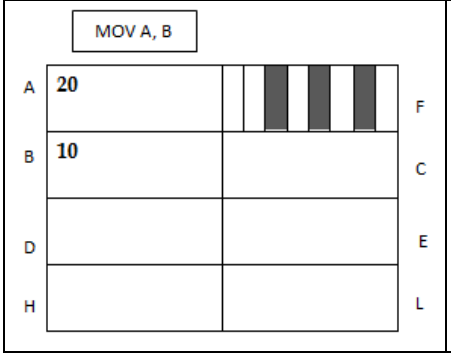
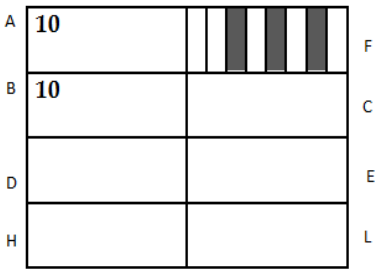
This group of instructions copies data from a location called source to another location called a destination, without modifying the value at the source. One of the primary functions of the microprocessor is coping data, from a register or input/ output or memory called the source, to another register or input/ output or memory called the destination.

Data Transfer	Example
Between registers	Register B → Register D
Specific data byte to register or memory	Data byte to → Register B
Between memory location and register	Memory location→ Register A
Between an I/O device and Accumulator.	Input device → Register A
Between a Register pair and the Stack	Register pair data → Stack Location

- MOV R_d, R_s (Copy data of one register to another register): MOV is an abbreviation of MOVE.
  - This instruction copies data from R_s source register to R_d destination register.
  - The example of R_s & R_d are all general-purpose registers such as A, B, C, D…
  - The contents of source register are not altered.

Operation: $R_s \rightarrow R_d$; No. of Bytes: 1 Byte; Flags: No flags are modified.

Example: MOV A, B; the content of B Reg. is copied to A Reg. Suppose A= 20, B= 10 and instruction MOV A, B is executed.

Before Execution	After Execution
	

Examples:

MOV B,	B	C	D	E	H	L	A
MOV C,	B	C	D	E	H	L	A
MOV D,	B	C	D	E	H	L	A
MOV E,	B	C	D	E	H	L	A
MOV H,	B	C	D	E	H	L	A
MOV L,	B	C	D	E	H	L	A
MOV A,	B	C	D	E	H	L	A

- MOV R, M or MOV M, R (Copies data of memory location to register): MOV R, M; this instruction copies data from memory M to register R.
  - The term M specifies the HL memory pointer.
  - The content of HL register pair is used as address and the content of that memory location is transferred to R i.e. register.
  - Examples of R, all general-purpose register.

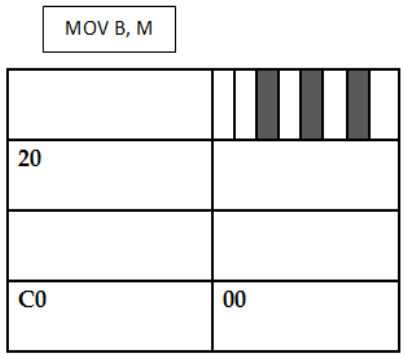
Operation: M →R or (HL) →R; No. of Byte: 1 Byte

Flags: No flags are modified.

Note: Whenever M term comes in any instruction it is memory pointer and the address will be given by HL pair. Bracket around HL specifies that contents are used as address.

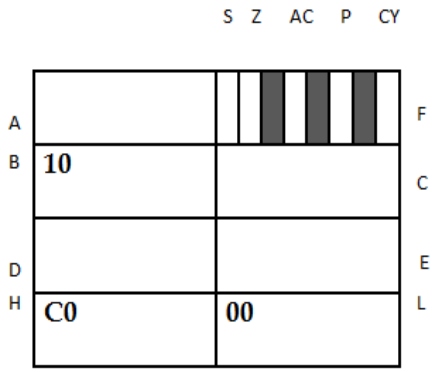
Example: MOV B, M; Suppose the contents of HL pair are C000H, B Reg. = 20H, at address C000H: 10H is stored and instruction MOV B, M is executed.

Before Execution



Address	Data
BFFF	XX
C000	10
C001	XX

After Execution



Address	Data
BFFF	XX
C000	10
C001	XX

* In instruction MOV B, M the data is transferred from memory to B reg. The HL register pair contents are used as address i.e. HL = C000H. The contents of memory location C000H are transferred to B reg. so, B reg. content will change from 20H to 10H.

Examples:

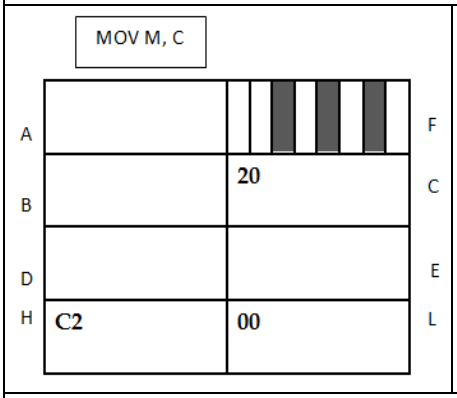
MOV B, M; MOV C, M; MOV D, M; MOV E, M; MOV H, M etc.

- MOV M, R (copies data of register to memory location): This instruction copies data from register R to memory M.
  - The term M specifies the HL memory pointer. The content of HL register pair is used as address and the content of specified register is transferred to that memory location.

Operation: R →M or R →(HL); No. of Bytes: 1 Byte; Flags: No flags are modified.

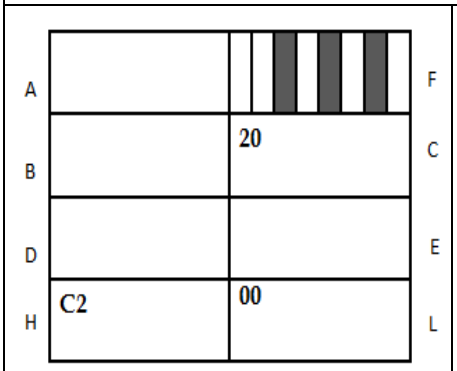
Example: MOV M, C Suppose the contents of HL pair are C200H, C reg. = 20H, at address C200H: 10H is stored and instruction MOV M, C is executed.

Before Execution



Address	Data
C1FF	XX
C200	10
C201	XX

After Execution



Address	Data
C1FF	XX
C200	20
C201	XX

- In instruction MOV M, C the data is transferred from B reg. to memory. The HL register pair contents are used as address i.e. HL = C200H.
- The content of register C is transferred to memory location C200H, so memory location C200H content will change from 10H to 20H.

Examples:

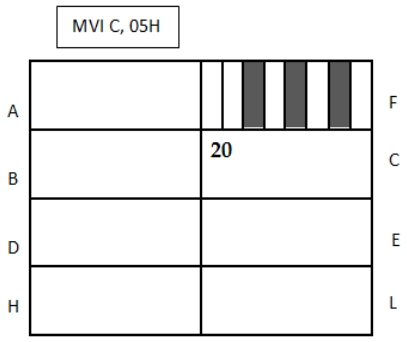
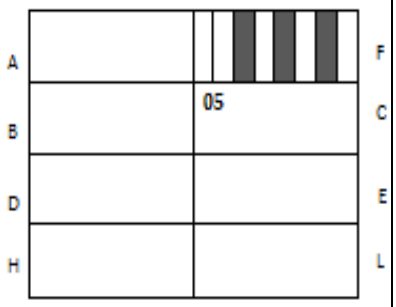
MOV M, B; MOV M, C; MOV M, D; MOV M, E; MOV M, H etc.

- MVI R, Data (Moves immediate data to specified register): This instruction moves immediate data to specified register.
  - The 8-bit data is specified with the instruction. The instruction moves the immediate 8-bit data to register. It is two - byte instruction, so first byte of instruction will be opcode and second byte will be 8-bit data.
  - The examples of R, all general-purpose registers.

Operation: Data →R; No. of Byte: 2 Bytes;

Flags: No flags are modified.

Example: MVI C, 05H; The data 05H will be loaded in C reg.

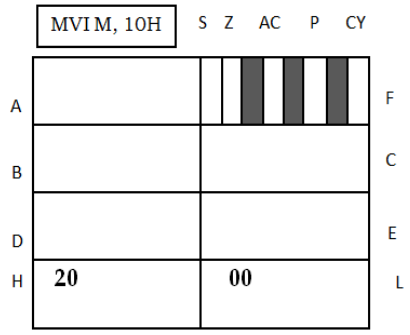
Before Execution	After Execution
	

- MVI M, Data (Moves immediate data to memory location): This instruction moves immediate data to memory.
  - The HL register pair is used as memory pointer. The contents of HL register pair are used as memory address and the immediate data is transferred to that memory location.

Operation: Data →M OR Data → (HL); No. of Byte: 2 Bytes; Flags: No flags are modified.

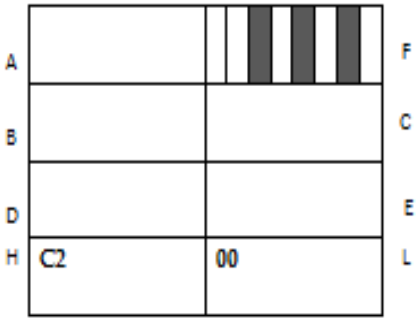
Example: MVI H, C2H; MVI L, 00H; MVI M, 10H; When MVI M,10 instruction is executed, the data 10H will be stored in memory location addressed by HL pair i.e. C200H

Before Execution



Address	Data
C1FF	XX
C200	10
C201	XX

After Execution

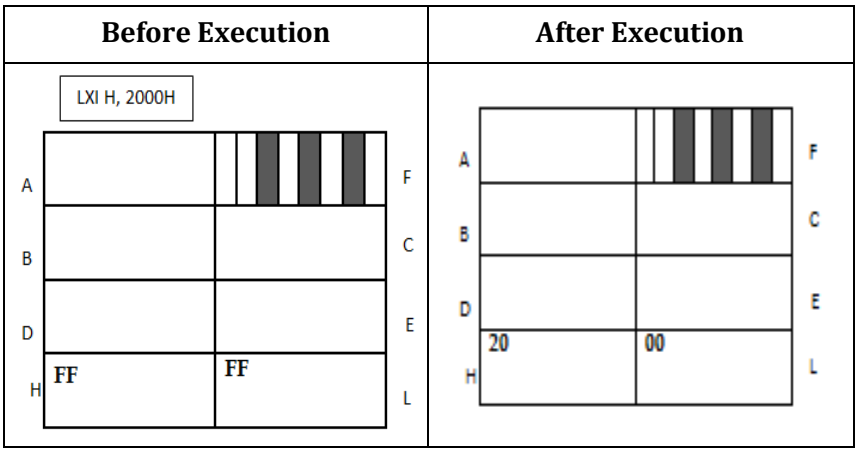


Address	Data
C1FF	XX
C200	20
C201	XX

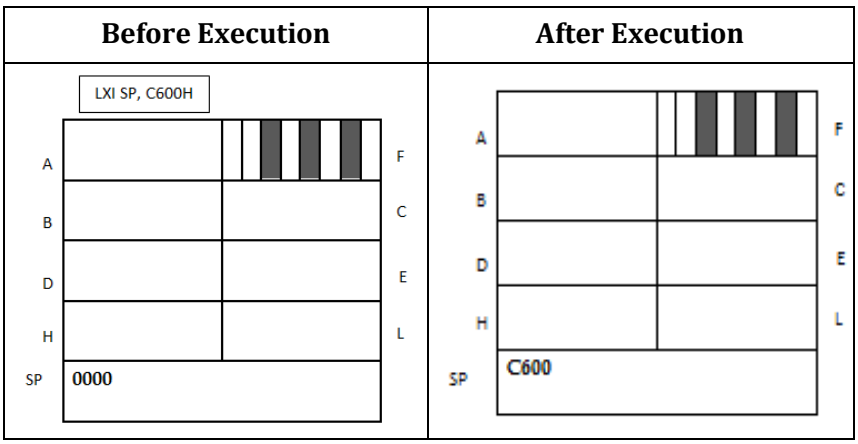
- LXI R_p, Data 16-bit (Load registers pair with 16-bit data): This instruction loads 16-bit data specified with instruction to the R_p register pair.
  - In instruction only high order register is specified for register pair, i.e. if HL pair is loaded only H reg. will be specified in the instruction.
  - The examples of R_p are BC pair, DE pair, HL pair and SP (Stack Pointer).

Operation: 16-bit Data $\rightarrow R_p$; No. of Byte: 3 Bytes; Flags: No flags are modified.

Example: LXI H, 2000H Load HL pair with data 2000H. 20H will be loaded in H reg. and 00H in L reg.



LXI SP, C600H (Load Stack Pointer with data C600H).



It is a 3-bytes instruction, so it is stored in memory in following format. First byte is opcode of instruction, second byte is low order 8-bits of data and third byte is high order 8-bits of data. This instruction can be used instead of 2 move instructions, for loading 2 registers of a register pair. For example, MVI B, 8-bit data; MVI C, 8-bit data, 2 move instructions are used to load register pair, instead you can use LXI B, 16-bit data.

Examples: LXI B, 16-bit data; LXI D, 16-bit data; LXI H, 16-bit data; LXI SP etc.

- LDA Address (Load accumulator direct from memory): This instruction copies the contents of the memory location to the accumulator.
  - The address of memory location is specified along with the instruction.
  - It is a 3 bytes instruction. So, it is stored in memory similar to LXI R_p, data instruction.

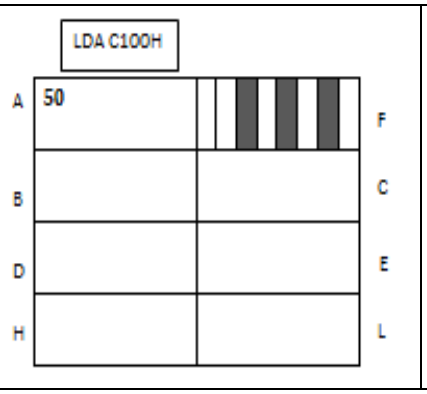
Operation: (Address) →A; No. of Byte: 3 Bytes;

Flags: No flags are modified.

Example: LDA C100H

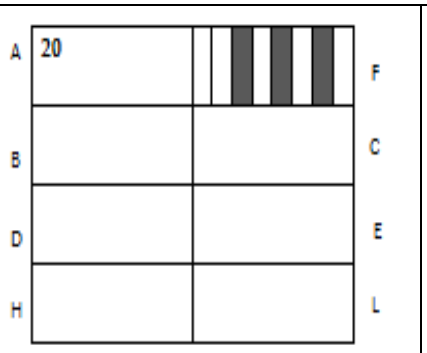
Load accumulator with the contents of memory location C100H.

Before Execution



Address	Data
C0FF	XX
C100	20
C101	XX

After Execution



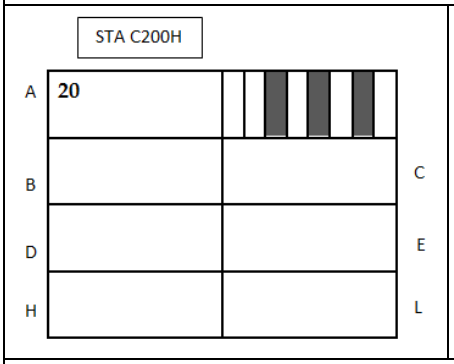
Address	Data
C0FF	XX
C100	20
C101	XX

- STA Address (Store accumulator direct to memory): This instruction copies the contents of the accumulator to the memory location.
  - The address of memory location is specified along with the instruction.
  - It is a 3-byte instruction.

Operation: A →(Address); Flags: No flags are modified.

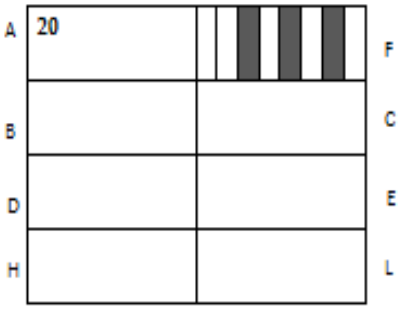
Example: STA C200H. Store the [A] at memory location C200H.

Before Execution



Address	Data
C1FF	XX
C200	10
C201	XX

After Execution



Address	Data
C1FF	XX
C200	20
C201	XX

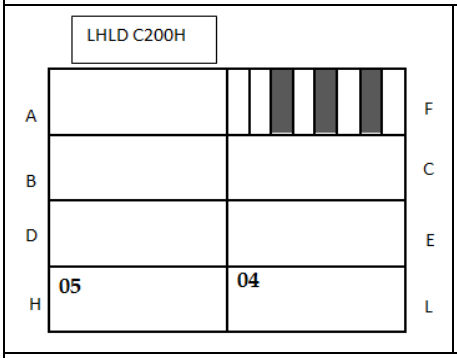
- LHLD Address (Load HL pair direct from memory): This instruction copies the content of the memory locations to H and L registers.
  - The address of memory location is specified along with the instruction. The content of memory location whose address is specified with the instruction is transferred to L reg. and (address+1) content to H reg.

- This instruction is used to load H and L registers from memory.
- It is a 3-byte instruction. So, its storing format is OPCODE as first byte, lower order address as second byte and high order address as third byte.

Operation: (Address) → L reg. and (Address+1) → H reg; Flags: No flags are modified.

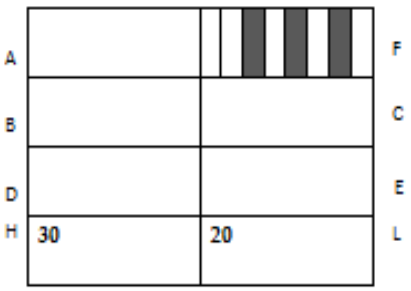
Example: LHLD C200H. Load HL pair from memory locations C200H and C201H. Suppose H = 05H and L = 04H, at memory locations C200H & C201H the data 20H and 30H is stored respectively and instruction LHLD C200H is executed.

Before Execution



Address	Data
C1FF	XX
C200	20
C201	30

After Execution



Address	Data
C1FF	XX
C200	20
C201	30

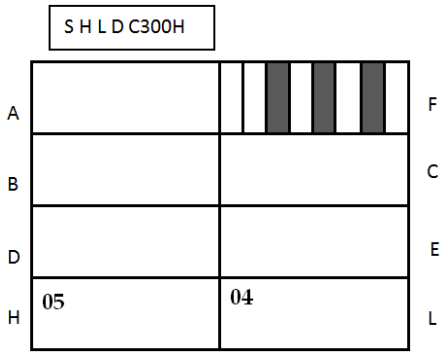
- SHLD Address (Store HL pair direct in memory): This instruction copies the content of the register H and L to the memory locations.

- The address of memory location is specified along with the instruction. The content of L reg. is stored at the memory location whose address is specified with instruction and the content of H reg. to the (address+1) location.
- This instruction is used to store H and L registers direct to memory.
- It is a 3-byte instruction.
- So, its storing format is opcode as first byte, lower order address as second byte and high order address as third byte.

Operation: L reg. →(Address)& H reg. →(Address+1); Flags: No flags are modified.

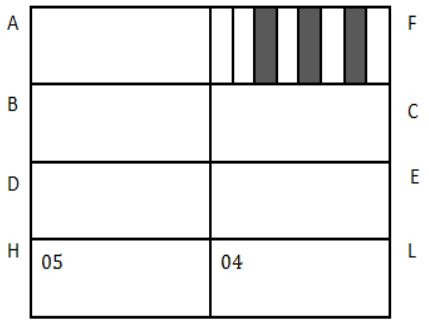
Example: SHLD C300H. Store the HL pair to the memory locations C300H and C301H.

Before Execution



Address	Data
C2FF	XX
C300	20
C301	30

After Execution



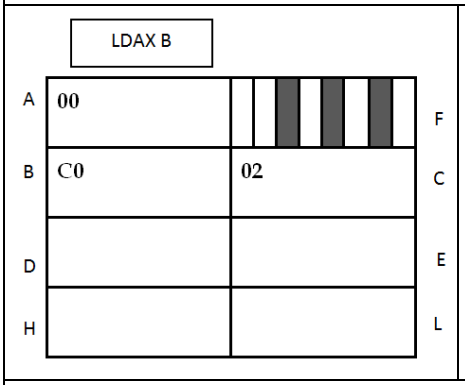
Address	Data
C2FF	XX
C300	04
C301	05

- LDAX R_p (Load accumulator indirect by using a memory pointer): This instruction copies the contents of the memory location to the accumulator.
  - The address of the memory location is given by the R_p (register pair) specified along with the instruction.
  - The examples of R_p are B (i.e. BC pair) and D (i.e. DE pair) only.

Operation: $(R_p) \rightarrow A$; No. of Byte: 1 Byte; Flags: No flags are modified.

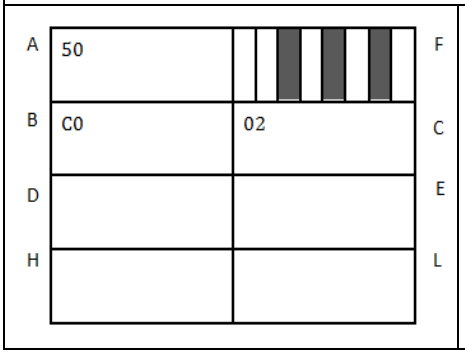
Example: LDAX B; Load accumulator with the content of memory location whose address is given by the BC register pair. Suppose A = 00H, B = C0H, C = 02H at memory location C002H: 50H is stored and instruction LDAX B is executed.

Before Execution



Address	Data
C001	XX
C002	50
C003	XX

After Execution



Address	Data
C001	XX
C002	50
C003	XX

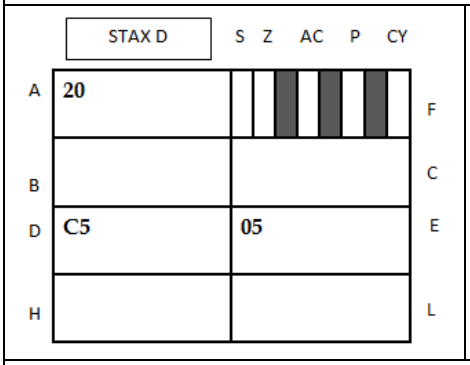
Examples: LDAX B and LDAX D.

- STAX R_p (Store accumulator indirect by using a memory pointer): This instruction copies the contents of the accumulator to memory location.
  - The address of the memory location is given by the R_p (register pair) specified along with the instruction.
  - The examples of R_p are B (i.e. BC pair) and D (i.e. DE pair) only.

Operation: A →(R_p); No. of Byte: 1 Byte; Flags: No flags are modified.

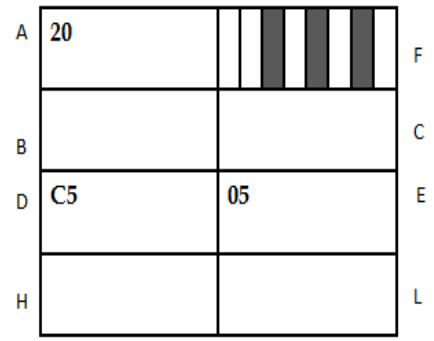
Example: STAX D; Store accumulator to the content memory location whose address is given by the DE register pair. Suppose A = 20H, D = C5H, E = 05H at memory location C505H: 10H is stored and instruction STAX D is executed.

Before Execution



Address	Data
C504	XX
C505	10
C506	XX

After Execution



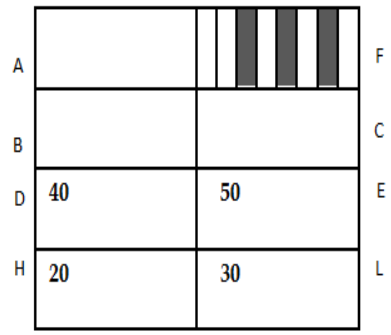
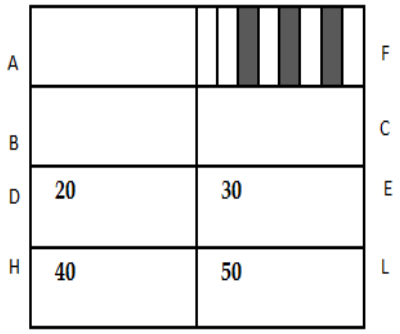
Address	Data
C504	XX
C505	20
C506	XX

Examples: STAX B and STAX D.

- XCHG (Exchange the content of HL register pair with DE register pair): This instruction exchanges contents of H reg. with D reg. and L reg. with E reg.

Operation: H ↔ D, L ↔ E; No. of Byte: 1 Byte; Flags: No flags are modified.

Example: XCHG; Suppose H = 20H, L = 30H, D = 40H, E = 50h and instruction XCHG is executed

Before Execution	After Execution
	

- NOP (No Operation): When this instruction is executed no operation is performed.

- The instruction is fetched and decoded, no operation is executed and microprocessor will go for the next instruction after that.
- This instruction can used as a small delay of 4T-states or if you wish to delete any instruction from your program you can use NOP instead of that instruction.

Operation: PC + 1 →PC; No. of Byte: 1 Byte; Flags: No flags are modified.

- HLT (Halt and enter wait state): When this instruction is executed, the microprocessor completes the execution and halts any further execution.
  - The microprocessor enters the halt acknowledge machine cycle and wait states are inserted in every clock period.

- The address and the data bus are placed in high impedance state. No. of Byte: 1 Byte
- Flags: No flags are modified.

➢ OUT 8-bit port address (Output to port): Send/copy the contents of the accumulator (A) to the output port specified in the second byte.
- No. of Byte: 2 Byte;
- Flags: No flags are modified.

➢ IN 8-bit port address (Input from port): Accepts/reads data from the input port specified in the second byte and loads into the accumulator.
- No. of Byte: 2 Byte;
- Flags: No flags are modified

C) Arithmetic Operations

Addition: Any 8-bit number or the content of a register or the content of a memory location can be added to the content of the accumulator and the sum is stored in the accumulator.

➢ ADD R (Add register R content to accumulator): This instruction adds the content of register R and accumulator and stores the result in accumulator.
- The examples of R, all general-purpose registers. In addition to the result in accumulator all flags are modified to reflect the result of operation.

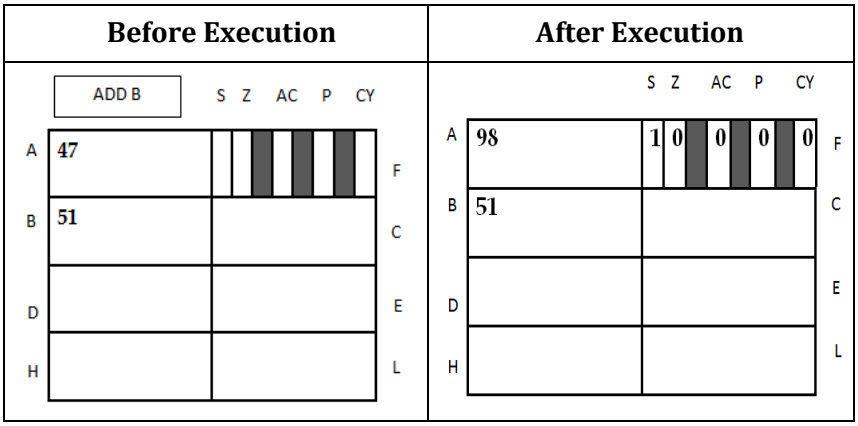
Operation: A + R →A; No. of Byte: 1 Byte Flags: All flags are modified.

Example: ADD B - (A + B → A); Suppose A = 47H, B = 57H and instruction ADD B is executed

A	0100 0111	=	47
B	0101 0001	=	51
--	----------	--	---
A	1001 1000	=	98

The addition is performed in hexadecimal number system and the flag status will be as follows: Zero flag is reset as result is not

equal to zero. Parity flag is reset as result contains odd number of 1s. Sign flag is set as D_7 bit of the result is 1. Carry flag is reset as there is no carry from D_7 bit of the result to next stage i.e. result is 8-bit only. The AC flag is reset as there is no carry from D_3 bit to D_4 bit of the result.



Examples: ADD B, ADD C, ADD D, ADD E, ADD H, ADD L, ADD A.

- DAA (Decimal Adjust Accumulator): The contents of the accumulator are changed from a binary value to its equivalent two, 4-bit binary decimal i.e. BCD number.
  - This is the only instruction that uses auxiliary carry flag to perform the operation of binary to BCD conversion.
  - DAA instruction working: The instruction checks the following condition

a. If the value of the low orders 4-bits D_3 – D_0 in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low order 4-bits of the accumulator.

b. If the value of the high orders 4-bits D_7 – D_4 in the accumulator is greater than 9 or if CY flag is set, the instruction adds 6 to the high order 4-bits of accumulator. Note: The restriction on DAA is condition of flags. If you execute any instruction which affects the flags CY and AC and then if you try to adjust accumulator to BCD format the result will be wrong as instruction will use AC and CY flags of previous instructions to take decisions. So DAA instruction is used with add instructions ADD, ADI etc. to perform addition

of number in BCD. The add instruction adds the two BCD numbers in hexadecimal format and DAA instruction converts this hexadecimal result to BCD format.

Operation: A reg. in binary →A reg. in BCD; No. of Byte: 1 Bytes;

Flags: All flags are modified.

Example:

1. If you want to add two BCD numbers 12 and 39 and want result in BCD form. The following steps should be implemented. MVI A, 12; ADI 39; DAA; Immediate data 39 is added to A (=12)

	0001 0010	=	12
+	0011 1001	=	39
	———————		——
	0100 1011	=	4B

The answer is 4B in hexadecimal form (without executing DAA instruction).

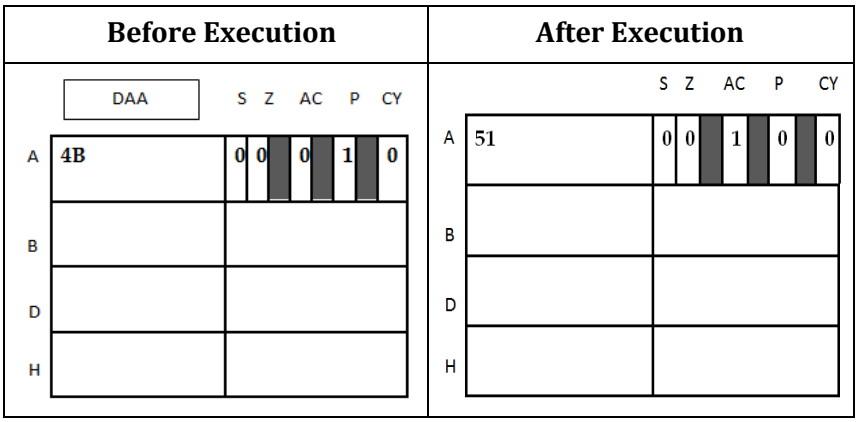
When DAA instruction is executed it checks:

a. Is low order 4-bits $D_3 - D_0$ greater than 9 or is AC flag is set. If any of the above condition is satisfied then 6 is added to low order 4-bits. In this example, B > 9 and AC = Reset so 6 will be added to low order 4-bits.

b. Is high order 4-bits $D_7 - D_4$ greater than 9 or is CY flag is set. If any of the above condition is satisfied then 6 is added to high order 4-bits. In this example, none of the condition is satisfied as 4 < 9 and CY = Reset so 6 is not added to high order 4-bits.

	0100 1011	=	4B
+	0000 0110	=	06
	———————		——
	0101 0001	=	51

Result in accumulator = 51 in BCD form.



2. If you want to add two BCD numbers 9 and 8 and want result in BCD form. The following steps should be implemented.

MVI A, 09

MVI B, 08

ADD B

DAA

	0000 1001	=	09
+	0000 1000	=	08
	————		
	0001 0001	=	11

The answer 11 is not correct as addition operation has been performed in hexadecimal number system. So, we get A = 11 and flag reg. = 00X1X1X0. Now, the DAA instruction is executed. As AC flag = set, 6 is added to lower order 4-bits.

	0001 0001	=	11
+	0000 0110	=	06
	————		——
	0001 0111	=	17

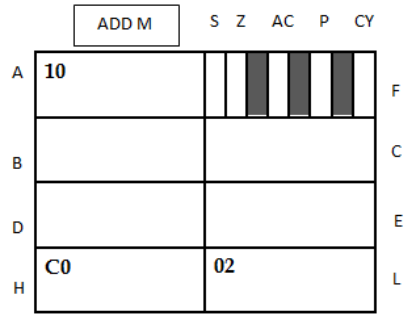
- ADD M (Add memory location content to accumulator): This instruction uses HL pair as memory pointer.
  - The content of memory location addressed by HL pair is added with accumulator and the result is stored in accumulator.

Operation: A + M →A OR A + (HL) → A; No. of Byte: 1 Byte; Flags: All flags are modified.

Example: ADD M - [A + (HL) → A]

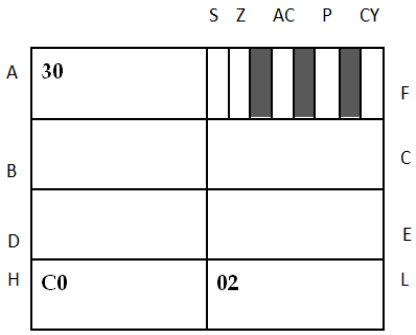
Suppose A = 10H, H = C0H, L = 02H at memory location C002H: 20H data is stored and ADD M instruction is executed. i.e. [10 + 20 → 30]

Before Execution



Address	Data
C000	XX
C001	XX
C002	20

After Execution



Address	Data
C000	XX
C001	XX
C002	20

- ADC R (Add register R and carry flag contents to accumulator): This instruction adds the contents of register R, carry flag CY and accumulator and stores the result in accumulator.
  - The examples of R, all general-purpose registers.
  - In addition to result in accumulator all flags are also modified to reflect the result of operation.

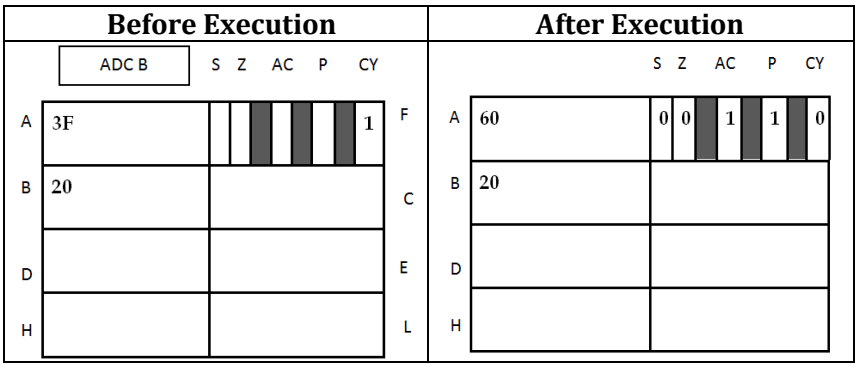
Operation: A + R + CY →A; No. of Byte: 1 Byte; Flags: All flags are modified to reflect the result of addition.

Example: ADC B - (A + B + CY → A)

Suppose A = $3F_H$, B = 20_H, CY = Set and instruction ADC B is executed.

A	0011	1111
B	0010	0000
CY		0001
—	———	———
A	0110	0000

So, CY = Reset, Z = Reset, P = Set, AC = Set, S = Reset. The previous carry flag will be reset because addition has not given a carry.



Examples: ADC B, ADC C, ADC D, ADC E, ADC H, ADC L, ADC A.

- ADC M (Add memory location and carry flag contents to accumulator): This instruction adds the contents of register R, carry flag CY and accumulator and stores the result in accumulator.
  - In addition to result in accumulator all flags are also modified to reflect the result of operation.

Operation: A + (HL) + CY →A OR A + M + CY → A; No. of Byte: 1 Byte;

Flags: All flags are modified.

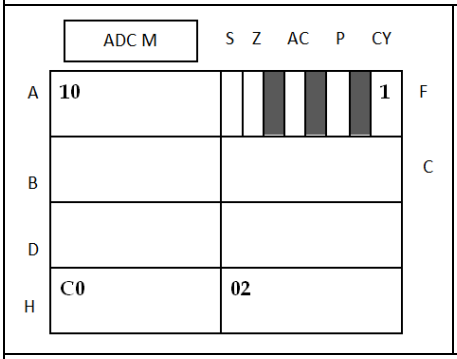
Example: ADC M - (A + (HL) + CY → A)

Suppose A = 10H, H = C0H, L = 02H, CY = Set at the memory location C002H: 20H data is stored and instruction ADC M is executed.

A	0001	0000
M	0010	0000
CY	0000	0001
___	____	_____
A	0011	0001

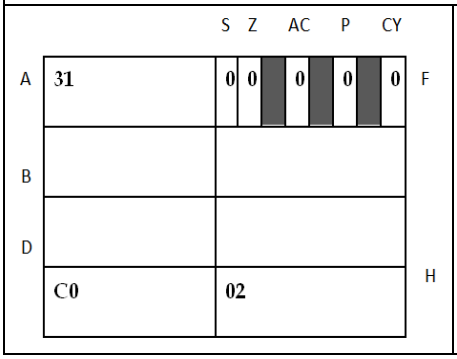
So, CY = Reset, Z = Reset, P = Reset, AC = Reset, S = Reset. The previous carry flag will be reset because addition has not given a carry.

Before Execution



Address	Data
C000	XX
C001	XX
C002	20

After Execution



Address	Data
C000	XX
C001	XX
C002	20

- ADI Data (Add immediate data to accumulator): This instruction adds the 8-bits of data specified along with the instruction to accumulator and result is stored in accumulator.
  - The storing format of this instruction will be first byte OPCODE and second byte operand (data).

Operation: A + data →A; No. of Byte: 2 Bytes;

Flags: All flags are modified.

Example: ADI B7H - (A + data → A)

Suppose A = 59 and instruction ADI B7H is executed.

A	0101	1001
DATA	1011	0111
———	———	———
A	1 0001	0000

So, CY = Set, Z = Reset, P = Reset, AC = Set, S = Reset.

- DAD R_p (Add the specified register pair to HL pair): This instruction adds the contents of specified register pair to HL pair and stores the result in HL pair. The examples of R_p are SP, BC, DE and HL.

Operation: R_p + HL → HL; No. of Byte: 1 Byte;

Flags: Only carry flag is modified

Example: DAD D - (DE + HL → HL); Suppose D = 20H, E = 35H, H = 80H, L = 45H, flag register = 10X1X0X1 and instruction DAD D is executed.

DE	2035
HL	8045
———	———
HL	A07A

So, CY flag will be reset and HL will contain A07AH as a result.

Example: DAD B, DAD D, DAD H and DAD SP.

Subtraction: Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the result is stored in the accumulator. The subtraction is performed in 2's complement and the results if negative is expressed in 2's complement. No two other registers can be subtracted directly. Operations of subtractions are as similar as addition above.

Examples: SUB R, SBB R, SUI Data, etc……..

Increment/Decrement: The 8-bit contents of a register or a memory location can be incremented or decremented by 1.

- Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decremented by 1.
- These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

➢ INR R (Increment registers contents by one): These instruction increments the content of the specified register by 1 and result is stored in the same register.

- The examples of R, all general-purpose registers.

Operation: R + 1 →R; No. of Byte: 1 Byte

Flags: Except carry flag, all other i.e. S, Z, P, AC flags are modified to reflect the result of operation.

Examples: INR B, INR C, INR D, INR E, INR H, INR L, INR A.

➢ INR M (Increment memory location contents by one): These instruction increments the content of memory location addressed by HL register pair by 1 and result is stored back at same memory location.

- Only carry flag is not modified, all other flags are modified.

Operation: (HL) + 1 →(HL) OR M + 1 = M.

No. of Byte: 1 Byte, Flags: Except carry flag, all other i.e. S, Z, P, AC flags are modified to reflect the result of operation.

- DCR R (Decrement the content of register by one): This instruction decrements the content of the specified register by 1 and result is stored in the same register.
  - The examples of R, all general-purpose registers.

Operation: R - 1 →R; No. of Byte: 1 Byte, Flags: Except carry flag, all other i.e. S, Z, P, AC flags are modified.

Examples: DCR B, DCR C, DCR D, DCR E, DCR H, DCR L, DCR A.

- INX R_p (Increment register pair content by one): These instruction increments the content of the specified register pair by 1 and result is stored in the same register pair.
  - The examples of R_p are all BC, DE, HL and SP. Only higher order register is specified for register pair in instruction.

Operation: R_p + 1 →R_p; No. of Byte: 1 Byte, Flags: No flags are modified.

Examples: INX B, INX D, INX H.

- DCX R_p (Decrement registers pair contents by one): This instruction decrements the content of the specified register pair by 1 and result is stored in the same register pair.
  - The examples of R_p are all BC, DE, HL and SP. Only higher order register is specified for register pair in instruction.

Operation: R_p - 1 →R_p; No. of Byte: 1 Byte; Flags: No flags are modified.

Example: DCX D – (DE - 1 →DE)

DCX H - (HL + 1 →HL); Suppose D = 02H, E = FFH, H = C2H, L = 00H, flag reg = 01X1X1X0 and instructions DCX D and DCX H are executed.

DE - 1 →DE; 02FF - 1 → 02FE; So, B = 02H and C = FEH.HL - 1 →HL; C200 - 1 → C1FF; So H = C1H and L = FF.

Examples: DCX B, DCX D, DCX H, DCX SP.

C) Logical Operations

These instructions perform various logical operations with the contents of the accumulator. Examples; AND, OR Exclusive-OR etc.

- ANA R: The content of the specified register is logically AND with the contents of accumulator and result is placed in accumulator.
  - The operation of AND is performed bit by bit i.e. D_0 of accumulator is AND with bit D_0 of register and so on up to D_7 bit of accumulator AND with D_7 bit of register.
  - Examples of R, all general-purpose register.

Operation: AND R →A No. of Byte: 1 Byte; Flags: S, Z, P are modified to reflect the result of operation. CY is reset and AC is set.

Example: ANA B

Suppose A = 56H, B = 82H and instruction ANA B is executed.

A	=	0101	0110
B	=	1000	0010
A	=	0000	0010

So, result in the accumulator will be 02H and flags status will be as follows:

CY = Reset, AC = Set, Z = Reset, S = Reset, P = Reset.

Before Execution

ANA B

		S	Z	AC	P	CY		
A	56						F	
B	82						C	
D							E	
H							L	

After Execution

		S	Z	AC	P	CY
A	02	0	0	1	0	0
B	82					
D						
H						

Examples: ANA B, ANA C, ANA D, ANA E, ANA H, ANA L, ANA A.

- ANA M: The content of the memory location is logically AND with the contents of accumulator and result is placed in accumulator.

- The operation of AND is performed bit by bit i.e. D_0 of accumulator is AND with bit D_0 of data at memory location and so on up to D_7 bit of accumulator AND with D_7 bit of memory location data.

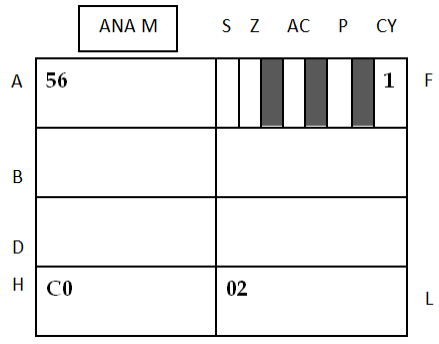
ANA M →A No. of Byte: 1 Byte; Flags: S, Z, P are modified to reflect the result of operation. CY is reset and AC is set.

Example: ANA M; Suppose A = 56H, M= 82H and instruction ANA M is executed.

A	=	0101	0110
M	=	1000	0010
—	—	———	———
A		0000	0010

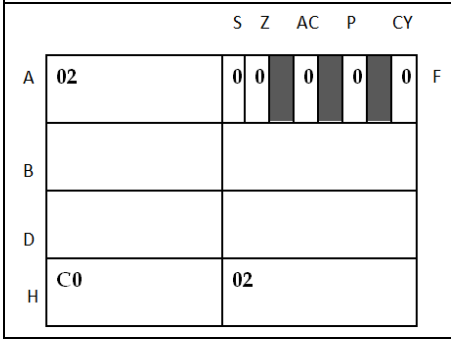
So, result in the accumulator will be 02H and flags status will be as follows:

Before Execution



Address	Data
C000	XX
C001	XX
C002	82

After Execution



Address	Data
C000	XX
C001	XX
C002	82

- ANI Data: The data is immediately AND with the content of accumulator and result is placed in accumulator.
  - The operation of AND is performed bit by bit i.e. D_0 of accumulator is AND with bit D_0 of data and so on up to D_7 bit of accumulator AND with D_7 bit of data.

Operation: A AND data →A; No. of Byte: 2 Bytes; Flags: All flags are modified.

Example: ANI 82H - (A AND data → A); Suppose A = 56 and instruction ANI 82H is executed.

A	0101	0110
DATA	1000	0010
———	———	———
A	0000	0010

After execution of the operation accumulator loaded by content 02H.

- ORA R: The content of the specified register is logically OR with the contents of accumulator and result is placed in accumulator.
  - The operation of OR is performed bit by bit i.e. D_0 of accumulator is OR with bit D_0 of register and so on up to D_7 bit of accumulator OR with D_7 bit of register.
  - Examples of R, all general-purpose register.

Operation: OR R →A No. of Byte: 1 Byte, Flags: S, Z, P are modified to reflect the result of operation. CY is reset and AC is set.

Example: ORA B; Suppose A = 56H, B = 82H and instruction ORA B is executed.

A	=	0101	0110
B	=	1000	0010
—		———	———
A	=	1101	0110

So, result in the accumulator will be D6H.

Before Execution							After Execution							
ORA B		S	Z	AC	P	CY			S	Z	AC	P	CY	
A	56						A	D6	0	0	1	0	0	
B	82						B	82						
D							D							
H							H							

- ORA M: The content of the memory location is logically OR with the contents of accumulator and result is placed in accumulator.
  - The operation of OR is performed bit by bit i.e. D_0 of accumulator is OR with bit D_0 of data at memory location and so on up to D_7 bit of accumulator OR with D_7 bit of memory location data.

OR M →A No. of Byte: 1 Byte Flags: S, Z, P are modified to reflect the result of operation. CY is reset and AC is set. Example: ORA M

Suppose A = 56H, M= 82H and instruction ANA M is executed.

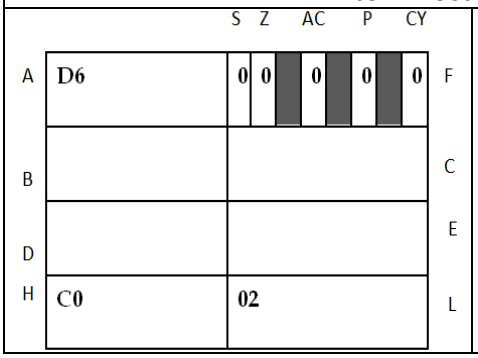
A	=	0101	0110
B	=	1000	0010
$\overline{\text{A}}$	=	$\overline{1101}$	$\overline{0110}$

So, result in the accumulator will be 02H and flags status will be as follows:

Before Execution																
ORA M		S	Z	AC	P	CY										
A	56					1	F									
B							C									
D																
H	C0	02														

Address	Data
C000	XX
C001	XX
C002	82

After Execution



Address	Data
C000	XX
C001	XX
C002	82

- ORI Data: The data is immediately OR with the content of accumulator and result is placed in accumulator.
- The operation of OR is performed bit by bit i.e. D_0 of accumulator is OR with bit D_0 of data and so on up to D_7 bit of accumulator OR with D_7 bit of data.

Operation: A OR data →A ; No. of Byte: 2 Bytes; Flags: All flags are modified.

Example: ORI 82H - (A OR data → A)

Suppose A = 56 and instruction ORI 82H is executed.

A	=	0101	0110
B	=	1000	0010
A	=	1101	0110

After the execution of the operation, accumulator loaded by content D6H.

Examples: XRA R, XRA M, XRI Data

These instructions are used to performs "Exclusive OR" operations with accumulator. The result of the operation is stored in accumulator.

- CMA (complement accumulator): This instruction complements the contents of accumulator and result is placed in accumulator.

- The complement is performing a NOT operation with each bit i.e. all 0's will be replaced by 1's and all 1's with 0's.

Operation: $\overline{\mathbf{A}} \rightarrow \mathbf{A}$(Bar placed above A is NOT operation); No. of Byte: 1 Byte

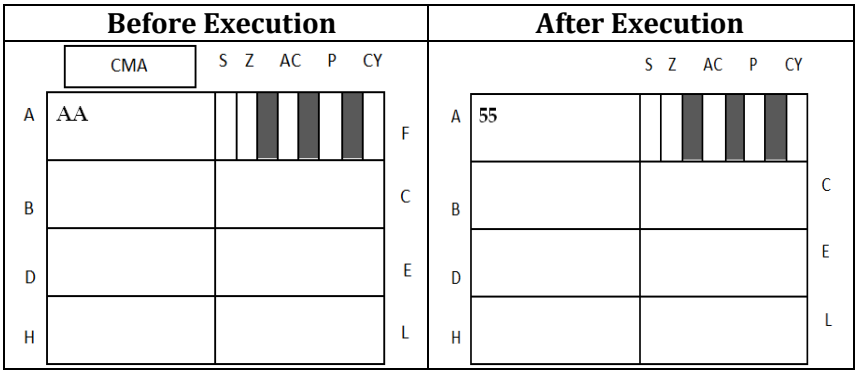
Flags: No flags are modified.

Example: CMA

Suppose A = AAH and instruction CMA is executed.

$\mathbf{A}$ = 1 0 1 0 1 0 1 0

$\overline{\mathbf{A}}$ = 0 1 0 1 0 1 0 1



- CMC (Complement the carry flag): This instruction complements the carry flag i.e. if carry flag = 1 instruction will Reset it and if carry flag = 0 instruction will Set it.

Operation: $\overline{\mathbf{CY}} \rightarrow \mathbf{CY}$; No. of Byte: 1 Byte

Flags: Only carry flag is complemented. No other flags are modified.

- STC (Set carry flag): This instruction sets the carry flag. It doesn't consider previous carry status. Only sets the carry flag. Operation: $\mathbf{CY} \rightarrow$ Set; No. of Byte: 1 Byte; Flags: Only carry flag is set. No other flags are modified.
- CMP R (Compare register with accumulator): This instruction compares the register contents with accumulator. The operation of comparing is performed by subtracting register content from the content of accumulator. The contents of

register & accumulator are not altered. The result of comparison is indicated by setting flags as follows:

[A] > [R]: CY flag is Reset & Z flag is Reset.

[A]= [R]: Z flag is Set.

[A] < [R]: CY flag is Set.

The S, P, AC flags are modified to reflect the result of subtraction and Z, CY are used to indicate the result of comparison.

Examples of R, all general-purpose registers.

Operation: A compare R → Flag register; No. of Byte: 1 Byte.

Flags: The S, P, AC flags are modified to reflect the result of subtraction and Z, CY are used to indicate the result of comparison

Example: CMP B

Suppose A = 20H and B = 10H and instruction CMP B is executed.

A = 0 0 1 0 0 0 0 0

B = 0 0 0 1 0 0 0 0

Results to: 0 0 0 1 0 0 0 0

Flag status will be as follows. CY = Reset, Z = Reset and P = Reset, S = Reset, AC = Reset. The A = 20H and B = 10H, contents of A and B are not altered, only result is indicated by carry flag & zero flag.

Examples: CMP B, CMP C, CMP D, CMP E, CMP H, CMP L and CMP A.

- CMP M (Compare content of memory location contents with content of accumulator): This instruction compares the memory location contents with accumulator. The address of memory location is given by HL reg. pair.
- The operation is performed in the same way as CMP R instruction. The result of comparison is indicated by setting flags same as in CMP R.
- Contents of accumulator and contents of memory location are not altered.

Operation: A compare M → Flag register; No. of Byte: 1 Byte

Flags: The S, P, AC flags are modified to reflect the result of subtraction and Z, CY are used to indicate the result of comparison.

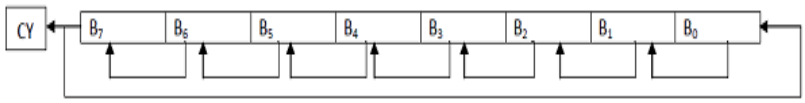
- CPI Data (Compare immediate data with the content of accumulator): This instruction compares the immediate data with accumulator.
  - The operation is performed in the same way as in CMP R instruction. The result of comparison is indicated by setting flags same way as in CMP R.

Operation: A compare data → Flag register; No. of Byte: 2 Byte

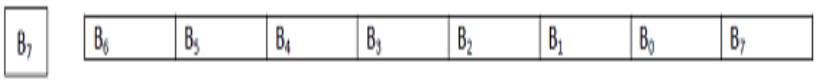
Flags: The S, P, AC flags are modified to reflect the result of subtraction and Z, CY are used to indicate the result of comparison.

- RLC: This instruction rotates the contents of accumulator to the left by one bit. It will shift B0 to B1, B1 to B2, ...B7 to B0 as well as to carry flag. Only CY flag is modified. It is a one-byte instruction. Implied addressing mode is used.

Before Execution



After Execution



Example: RLC

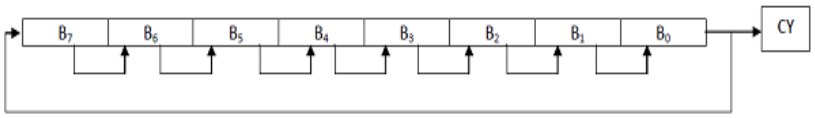
Suppose A = 1FH and instruction RLC is executed.

Before Execution	After Execution
RLC S Z AC P CY A 0001 1111 F	S Z AC P CY A 0011 1110 X X X X 0

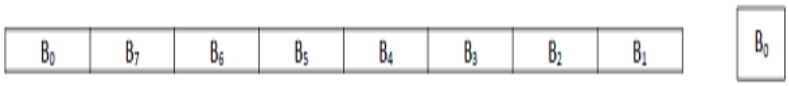
- RRC: This instruction rotates the contents of accumulator to the right by one bit. It will shift B7 to B6, B6 to B5, ...B0 to

B7 as well as to carry flag. Only CY flag is modified. It is a one-byte instruction. Implied addressing mode is used.

Before Execution



After Execution

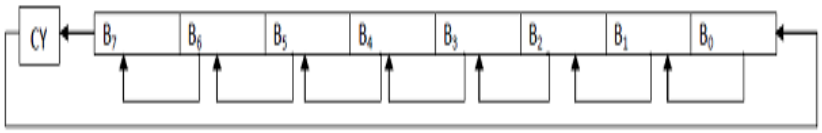


Example: RRC; Suppose A = 1CH and instruction RRC is executed.

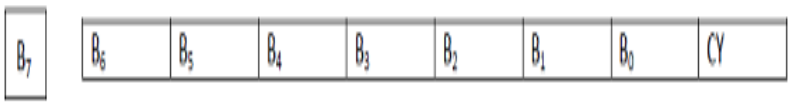
Before Execution	After Execution
RRC S Z AC P CY A 0001 1100 F	S Z AC P CY A 0000 1110 X X X X 0

- RAL: This instruction rotates the contents of accumulator to the left by one bit along with the carry. It will shift B0 to B1, B1 to B2, ...B7 to CY and CY to B0. Only CY flag is modified. It is a one-byte instruction. Implied addressing mode is used.

Before Execution

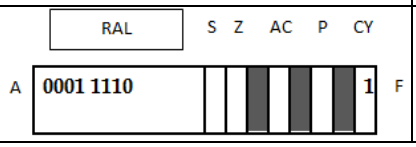
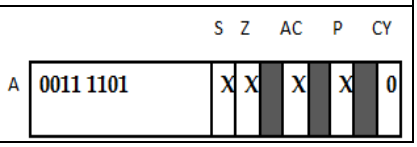


After Execution



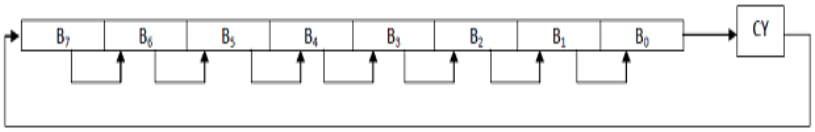
Example: RAL

Suppose A = 1EH and CY = Set and instruction RAL is executed.

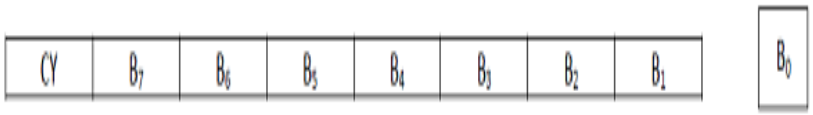
Before Execution	After Execution
	

- RAR: This instruction rotates the contents of accumulator to the right by one bit along with the carry. It will shift B7 to B6, B6 to B5, ...B0 to CY and CY to B7. Only CY flag is modified. It is a one-byte instruction. Implied addressing mode is used.

Before Execution

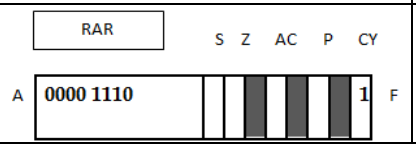
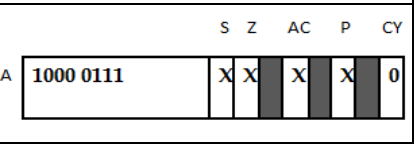


After Execution



Example: RAR

Suppose A = 0EH and CY = Set and instruction RAR is executed.

Before Execution	After Execution
	

D) Branching Operations

The microprocessor executes machine codes in a sequential manner. It goes on executing from one memory location to the next. Branch group of instructions, instructs the microprocessor to go to a different memory location. The address of the new memory location is either specified in the instruction or supplied by the microprocessor or given by extra hardware. The branch group instructions are classified into three categories:

(A) Jump instructions:	(1) JMP address (3) PCHL	(2) Conditional jump instructions
(B) Call and return instructions:	(1) CALL address (3) RET	(2) Conditional call instruction (4) Conditional RET instructions.
(C) Restart instructions:	RST N	

1. **Jump Instructions:**

- JMP Address (Unconditional jump to the address specified along with the instruction):
  a. When this instruction is executed, the instruction directs microprocessor about jump and it provides the address where microprocessor has to transfer the control.
  b. The storing format is; OPCODE, lower order address and higher order address. Now the instructions from NEW address will be executed in sequence.

Operation: Address → PC; No. of Byte: 3 Bytes; Flags: No flags are modified.

Example: JMP C200H. Suppose the above instruction is stored at address C005 as follows:

C005 OPCODE C006 00

C007 C2

When this instruction is fetched and executed by microprocessor, it transfers control of address C200 and starts executing instructions from C200 onwards.

2. **Conditional Jump Instructions:**

a. In conditional jump instructions, when the condition is true or satisfied then only jump is made at the specified address.

b. If condition is false or not satisfied it will just check and proceed further to execute the next instruction after it. Different conditional jumps available are:

1. JC: Jump, if carry flag is set.
2. JNC: Jump, if carry flag is reset.

3. JZ: Jump, if zero flag is set.
4. JNZ: Jump, if zero flag is reset.
5. JP: Jump, if positive i.e. sign flag is reset.
6. JM: Jump, if minus or negative i.e. sign flag is set.
7. JPE: Jump, if parity even i.e. parity flag is set.
8. JPO: Jump, if parity odd i.e. parity flag is reset.

**Remember there is no jump on auxiliary carry flag.

Operation: If condition is true, address → PC; If false PC = PC + 3;

No. of Byte: 3 Bytes.

Flags: No flags are modified, only flags are checked.

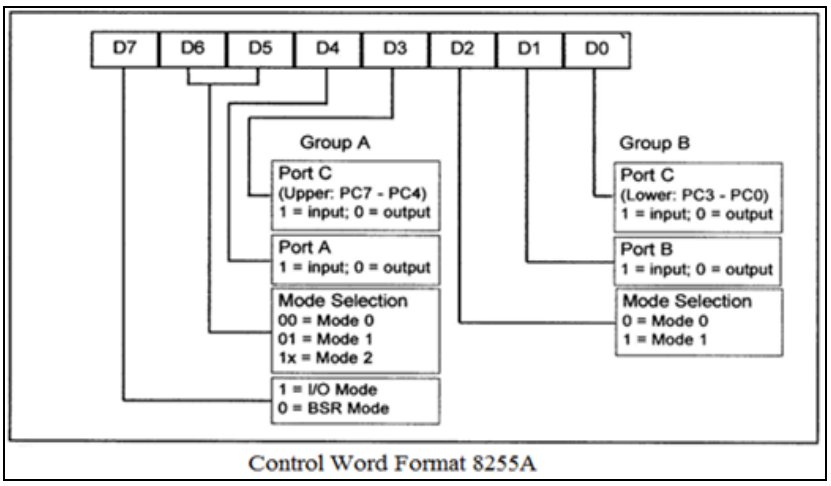
Example:

- JZ C200; This instruction checks zero flag.
  a. If zero flag is set, condition is true. So, program control is transferred to address C200.
  b. If zero flag is reset, condition is false. So, program control is not transferred instead it will execute the next instruction after JZ C200.
- JNZ C200
  a. This instruction checks zero flag.
  b. If zero flag is reset, condition is true. So, program control is transferred to address C200.
  c. If zero flag is set, condition is false. So, program control is not transferred instead it will execute the instruction after JNZ C200.

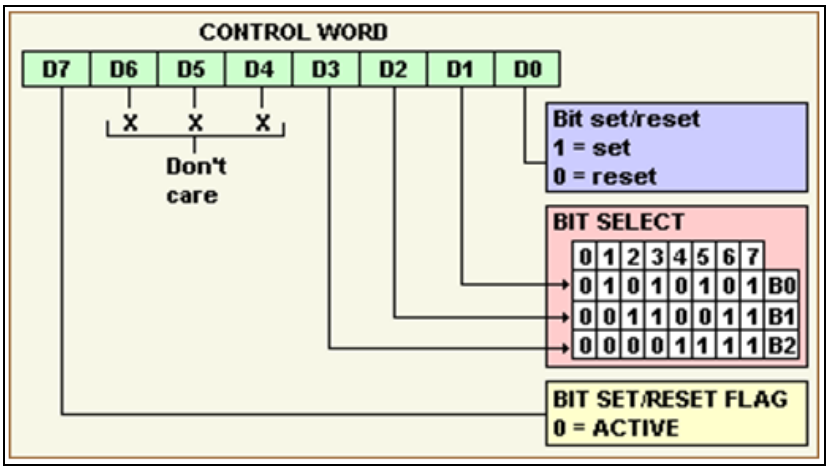
Chapter 4

PPI 8255A

Control Word 8255A as I/O Mode & BSR Mode:



Control Word Format 8255A



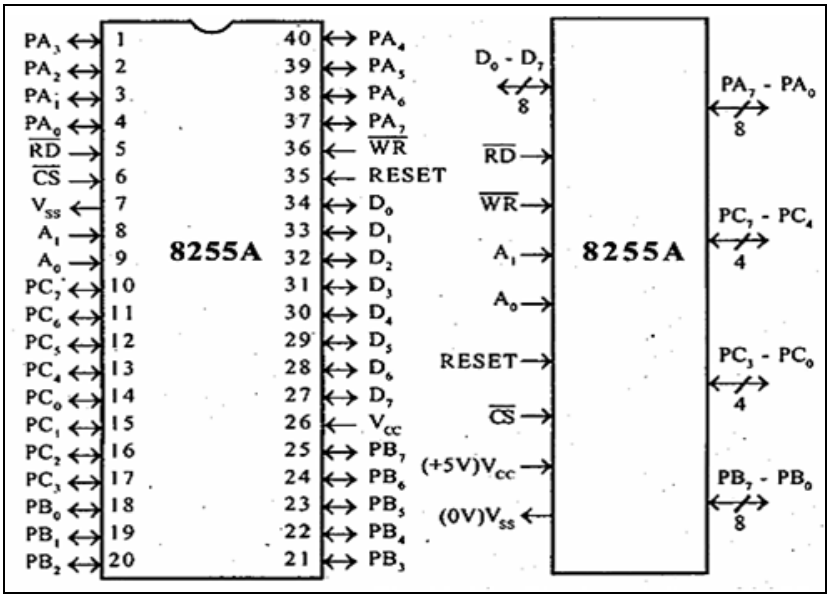
The 8255A is a general purpose programmable I/O device designed to transfer the data from simple I/O to interrupt I/O based on the conditions applied. It can be used with almost all Microprocessors & Microcontrollers. It consists of three 8-bit bidirectional I/O ports (24 I/O pins), can be configured in different Modes.

Ports of 8255 A:

8255 A has three ports, i.e., Port A, Port B & Port C.

- Port A contains one 8-bit output latch / buffer and one 8-bit input buffer.
- Port B is similar to Port A.
- Port C can be subdivided into Port C lower (PC0-PC3) and Port C upper (PC4-PC7) using appropriate control word.

These three ports are further divided into two groups, i.e. Group A & Group B. These two groups can be programmed in three different modes, named as Mode 0, Mode 1 & Mode 2.



Pin Diagram & Logic Diagram of PPI 8255

Operating Modes: 8255 A has three different operating modes

- Mode 0: In this mode, Port A and B is used as two 8-bit ports and Port C as two 4-bit ports. Each port can be programmed in either input mode or output mode where outputs are latched and inputs are not latched. Ports do not have interrupt capability.
- Mode 1: In this mode, Port A and B is used as 8-bit I/O ports. They can be configured as either input or output ports. Each

port uses three lines from port C as handshake signals. Inputs and outputs are latched.

- Mode 2: In this mode, Port A can be configured as the bidirectional port and Port B either in Mode 0 or Mode 1. Port A uses five signals from Port C as handshake signals for data transfer. The remaining three signals from Port C can be used either as simple I/O or as handshake for port B.

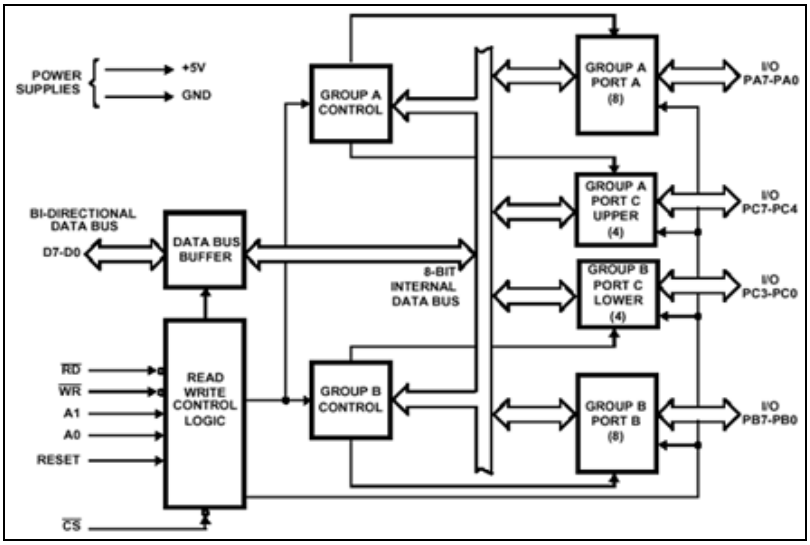
Features of 8255 A: The prominent features of 8255A are as follows:

- It consists of 3, 8-bit I/O ports i.e. PA, PB, and PC.
- Address/Data bus must be externally demultiplexed.
- TTL compatible.
- It has improved DC driving capability.

Stack Pointer Register:

1. The stack pointer is a sixteen-bit register, used to point at the stack.
2. Return addresses are stored in the stack.
3. In simple words stack acts like an auto decrement facility in the system.
4. The initialization of the stack done with the help of an instruction LXI SP.

8255 Architecture:



Subroutine in 8085: In computers, a subroutine is a sequence of program instructions that performs a specific task, packaged as a unit. This unit can then be used in programs wherever that particular task has to be performed. A subroutine is often coded so that it can be called several times and from several places during the execution of the program, including from other subroutines and once the subroutine's task is done, branch back (return) to the next instruction after the call, It is implemented by using Call and Return instructions. The different types of subroutine instructions are:

Unconditional Call Instruction: CALL address is the format for unconditional call instruction. After execution of this instruction program control is transferred to a sub-routine whose starting address is specified in the instruction. Value of PC (Program Counter) is transferred to the memory stack and value of SP (Stack Pointer) is decremented by 2.

INSTRUCTION	PARAMETER	COMMENT
CC	16-bit address	Call at address if CY (carry flag) = 1
CNC	16-bit address	Call at address if CY (carry flag) = 0
CZ	16-bit address	Call at address if ZF (zero flag) = 1
CNZ	16-bit address	Call at address if ZF (zero flag) = 0
CPE	16-bit address	Call at address if PF (parity flag) = 1
CPO	16-bit address	Call at address if PF (parity flag) = 0
CN	16-bit address	Call at address if SF (sign flag) = 1
CP	16-bit address	Call at address if SF (sign flag) = 0

Conditional Call Instruction: In these instructions program control is transferred to subroutine and value of PC is pushed into stack only if condition is satisfied.

Unconditional Return Instruction: RET is the instruction used to mark the end of sub-routine. It has no parameter. After execution of this instruction program control is transferred back to main program from where it had stopped. Value of PC

(Program Counter) is retrieved from the memory stack and value of SP (Stack Pointer) is incremented by 2.

Conditional Return Instruction: By these instructions program control is transferred back to main program and value of PC is popped from stack only if condition is satisfied. There is no parameter for return instruction.

Chapter 5

ASSEMBLY LANGUAGE PROGRAMMING

Monitor Subroutines @ Dyna 8085 Kit

Note: One should be familiar with MONITOR SUBROUTINES such as READ KEYBOARD i.e RDKBD (CD E7 02 located at 02 E7), DISPLAY on ADDRESS FIELD i.e MODIAD (CD 62 03 located at 03 62), DISPLAY on DATA FIELD i.e MODIDT (CD 6E 03 located at 03 6E) and DELAY (CD F1 05 located at 05 F1) mentioned in the 8085 μP kit's manual.

Understanding INSTRUCTION SET of 8085 & Study of 8085 microprocessor kit & commands through the questionnaire below:

1. Consider any arbitrary Hex number, write an ALP to transfer the number in all registers A, B, C, D, E, H & L. Verify after execution.
2. Consider any arbitrary Hex number, write an ALP to find its' 1's complement, 2's complement, predecessor & successor; store the number & results in some registers. Write an ALP to find negative of a number 6F and verify the same by a suitable method.
3. Write an ALP to add 10, 8-bit numbers located in consecutive memory locations & display the result in the data field using MS MODIDT.
4. Consider any two arbitrary Hex numbers, write an ALP to find their Addition, Subtraction, AND, OR, XOR and store the numbers & results in various registers.
5. Add two 16-bit Hex numbers using DAD command & use MS MODIAD to display the answer in the address field.
6. Multiply two 8-bit Hex numbers using a suitable ALP. Use MS MODIDT to display the answer in the data field.
7. Write an ALP to find the number of odd numbers from the given set of numbers. Modify the program to find even/positive/negative numbers.

Chapter 6

UNDERSTANDING BASICS

D	H	O	B
0	0	0	0000
1	1	1	0001
2	2	2	0010
3	3	3	0011
4	4	4	0100
5	5	5	0101
6	6	6	0110
7	7	7	0111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111

(A) Questions for Understanding:

1. Find the exact value of the number 1111 in all number systems (D/H/O).
2. Express the result of addition of 67 & 75 in all the number systems (D/H/O).
3. Find the Binary of $(8453)_H$ & $(7421)_O$.
4. $(1010101011110000 1000)_B = (X)_O = (Y)_H$. Find X & Y
5. Express the result of AND, OR & XOR of 4E & FC in Hexadecimal System.
6. Find: 05 x 07 = $(\ \ )_H$.
7. Determine the 2's complement of $(3C)_H$. Discuss it's significance.
8. Find the negative of the following numbers: 23, 45, 4D, 5C, 78, 9F, DD, C5.
9. Determine: 34 – D6 =?

10. Find: 4678 + FBCD + 3DEF =?
11. Find: 4DFCD – 3FFFF =?
12. Decode: 0001 1001 0100 0111.
13. Determine the Even/Odd/Positive/Negative numbers from the given set of numbers: 4A, 6B, 8C, 7D, E8, F5, DD, EF.

(B) Questions for Practice:

1. Find the exact value of the number 3475 in all number systems (D/H/O).
2. Express the result of addition of 897 & 878 in D/H number systems.
3. Find the Binary of $(7201)_H$ & $(6210)_O$.
4. $(10101110111100001111111110000111)_B = (X)_O = (Y)_H$. Find X & Y.
5. Express the result of AND, OR & XOR of 05 & 89 in hexadecimal system.
6. Find: D3 + FE = ($)_H$. Also, discuss the status of FLAG register.
7. Find: 08 x 09 = ($)_H$.
8. Determine the 2's complement of $(DC)_H$. Discuss it's significance.
9. Find the negative of the following numbers: DF, 4F, D6, 5F, F2, 9C, BB, CD.
10. Find: 3D – DF =?
11. Find: 46FF + F03D + 366F =? & 44679 – 3FD2D =?

Addition: MVI B, 25 MVI C, AF MOV A, B ADD C HLT	

A	25 + AF = D4		F
B	25	AF	C
D			E
H			L

(A) Questions for Understanding:

1. Modify the above program for subtraction.
2. Modify the programs of addition & subtraction such that the data is available in certain memory locations and the result should be also in successive memory location.
3. Write an ALP to find 2's complement of a number. Verify that the 2's complement of the number is actually negative of the number.
4. Write an ALP to add four 8-bit data. After the execution of the program, the result and the data, both should be visible to the programmer through the registers.
5. State the Flag register status when a larger number is subtracted from a smaller number.

(B) Questions for Practice:

1. Modify the program by using minimum number of bytes if result of addition is only required.
2. Modify the program for all logical operations (AND, OR, XOR).
3. Modify above question with minimum number of bytes.
4. Modify the program of subtraction by using minimum number of bytes if result of subtraction is only required.

Chapter 8

μP PROGRAMMING-II

Flags: The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags; they are listed in table in figure and their bit positions in the flag register are shown in figure (c). The most commonly used flags are Zero, Carry and Sign. The microprocessor uses these to test data conditions.

S	Z	X	AC	X	P	X	CY

The following flags are set or reset after the execution of an arithmetic or logic operation; data copy instructions do not affect any flags.

- Z; Zero: The Zero flag is set to 1 when the result is zero; otherwise it is reset.
- CY; Carry: If an arithmetic operation results in a carry, the CY flag is set; otherwise it is reset.
- S; Sign: The sign flag is set if bit D_7 of the result = 1; otherwise it is reset.
- P; Parity: If the result has an even number of 1s, the flag is set; for an odd number of 1s, the flag is reset.
- AC; Auxiliary Carry: In an arithmetic operation, when a carry is generated by digit D_3 and passed to digit D_4, the AC flag is set. This flag is used internally for BCD (binary-coded-decimal) operations; there is no Jump instruction associated with this flag.

(A) Questions for Understanding:

1. How register A differs from other registers?
2. Name the registers which can be used in a pair.
3. What is special about HL pair compared to BC & DE pair?

4. Which of the registers are not accessible to the programmer?
5. State the Flag Status if BC is added to DF.
6. What is the role of carry flag in the process of rotation of 8-bit data?
7. Find; FE + 8D = ( ) $_H$. Also, discuss the status of FLAG register.
8. What is the role of Z-bit, S-bit, P-bit, C-bit & A- bit in the Flag register?
9. Name few instructions which do not affect Flag Register.
10. Name few instructions who bypass the Accumulator.
11. What is the role of 16-bit PC & SP?
12. Differentiate the role of Carry Flag in RLC & RAL.

(B) Questions for Practice:

1. How register F differs from other registers?
2. Name the registers which cannot be used in a pair?
3. Which register pair is used for pointing memory location?
4. For which register pairs the exchange instruction is valid?
5. State the Flag Status if AE is added to CF.
6. What is the role of A in IN / OUT instructions?
7. Find; BE + 8F = ( ) $_H$. Also, discuss the status of FLAG register.
8. What is the meaning of X in the Flag Register?
9. Name at least 10 instructions which are possible only through Accumulator.
10. Which register pair will be preferred for 16- bit addition?

Chapter 9

μP PROGRAMMING-III

(A) Multiplication

```
XRA A
MVI B, 04
MVI C, 08
** ADD B
   DCR C
    JNZ  **
    HLT
```

(B) Data Transfer

```
LXI H C030
LXI D C050
MVI C, 05
** MOV A, M
   STAX D
   INX  H
   INX  D
   DCR C
   JNZ **
  HLT
```

(C) 16-Bit Addition

```
LHLD, C050
XCNG
LHLD, C052
DAD D
SHLD, C054
HLT
```

Concept of Even / Odd

- 35 0011 010 1 ODD
- 6A 0110 101 0 EVEN

Rotate the data right & check carry flag status

Concept of Positive / Negative

- D2 1 101 0010 NEGATIVE
- 4D 0 100 1101 POSITIVE

Rotate the data left & check carry flag status

(D) Number of Odd Numbers

```
LXI H, C050
MVI C, 00
* MOV A, M
  RAR
  JNC **
  INR C
** INX H
   MOV A, L
  CPI 5A
  JNZ *
  MOV A, C
  STA C800
  HLT
```

Chapter 10

DELAY

μP processes the data in micro seconds to whom it's slow peripherals cannot respond more over human eye cannot respond to such fast variations.

(A) To generate SMALL delay

```
        MVI B, 10
LOOP    DCR B
        JNZ  LOOP
        RET
```

(B) To generate BIG delay

```
        LXI D, FFFF
LOOP    DCX D
        MOV A, D
        ORA E
        JNZ LOOP
        RET
```

(C) To generate BIGGER delay

```
        MVI B, 10
LOOP I  MVI C, 78
LOOP II DCR C
        JNZ   LOOP II
        DCR B
        JNZ   LOOP I
        RET
```

Max DELAY up to 0.293 sec.

Chapter 11

WAVEFORM GENERATION

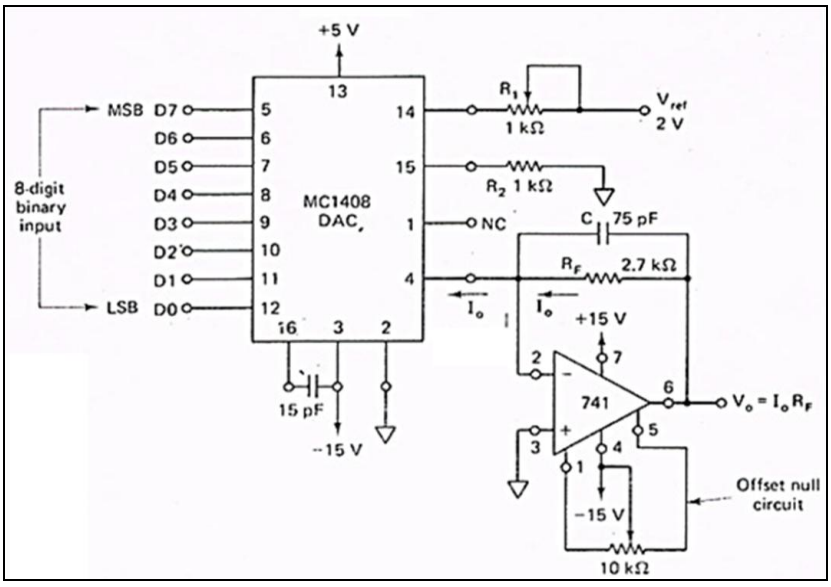
8255A is a popularly used parallel, programmable input-output device. It can be used to transfer data under various conditions from simple input-output to interrupt input-output. It is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.

Addresses:

Port A	$(80)_H$
Port B	$(81)_H$
Port C	$(82)_H$
Control Register	$(83)_H$

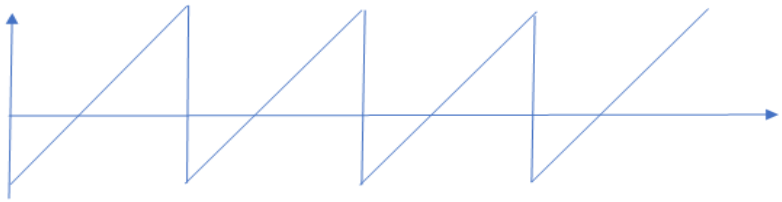
*** (Port addresses are different for different 8085 kits)

8-Bit DAC



Sawtooth Waveform Generator

The digital signals coming from µP through 8255, has to be converted into analog signal to see it on a CRO. This purpose of (remove) conversion is done by a DAC Circuit O/P Waveform:



Program:

Address	Label	Opcode	Mnemonics	Comment
	START	( )$_H$ ( )$_H$	MVI A (80)$_H$	Initialize Port A as O/P
		( )$_H$ ( )$_H$	OUT (83)$_H$	
	LOOP2	( )$_H$ ( )$_H$	MVI A (00)$_H$	Initialize the count
		( )$_H$ ( )$_H$	OUT (80)$_H$	Send count to Port A
	LOOP1	( )$_H$	INR A	Increase content of acc. by one
		( )$_H$ ( )$_H$	OUT (80)$_H$	Send count to Port A
		( )$_H$ ( )$_H$	CPI (FF)$_H$	Compare the [A] with data (FF)$_H$
		( )$_H$ ( )$_H$ ( )$_H$	JNZ LOOP1	Jump if comparison not equal to (FF)$_H$
		( )$_H$ ( )$_H$ ( )$_H$	JMP LOOP2	Jump if comparison is equal to (FF)$_H$

(A) Questions for Understanding:

1. Write a Control Word in BSR mode to SET PC.7?
2. Write a Control Word in BSR mode to RESET PC.0?
3. Write an ALP to RESET & SET PC.3 alternatively with an appropriate delay?
4. Write an ALP to blink an LED @ PC.6 with an appropriate delay?

5. Write a Control Word in I/O mode for all Ports of 8255 as I/P Ports.
6. Write a Control Word in I/O mode for Ports A & B as I/P Ports & Port C as an O/P Port.
7. Write various ALPs to blink 8-bit LEDs using µ8085 & PPI 8255 at appropriate delay.
8. What is the step size of an 8-bit DAC?

(B) Questions for Practice:

1. Write a Control Word in BSR mode to SET PC.5?
2. Write a Control Word in BSR mode to RESET PC.1?
3. Write an ALP to RESET & SET PC.7 alternatively with an appropriate delay?
4. Write an ALP to blink an LED @ PC.3 with an appropriate delay?
5. Write a Control Word in I/O mode for all Ports as O/P Ports.
6. Write a Control Word in I/O mode for Ports B & C as I/P Ports & Port A as an O/P Port.
7. Write an ALP to receive the data @ Port B & Port C & to display the difference at Port A.
8. Write an ALP to receive the data @ Port B & Port A & to display the OR ed at Port C.
9. Write an ALP to generate asymmetric square waveform.
10. Interfacing 8 switches and 8 LEDs to 8255:
    a. Write ALP to read the status of the switches and display on the LEDs.
    b. Write ALP so that when the first switch is made ON, all the LEDs should glow & when the second switch is made OFF, all the LEDs should be off.
11. Write an ALP to generate waveforms using µP 8085, PPI 8255 & a suitable DAC.
    a. Symmetric/Asymmetric Square
    b. Triangular
    c. Ramp Waveforms
    d. Trapezoid

Chapter 12

UNDERSTANDING PROGRAMMING

Experiment 1: Write an ALP to add two 8-bit numbers that are stored in two consecutive memory locations $(****)_H$ & $(**** + 1)_H$ respectively. Store the sum & carry in next consecutive memory locations.

Label	Mnemonics	Hex Codes	Comments
START	LXI H, (****) H		Load memory location of first number
	MOV B, M		Load a number in reg B
	INX H		Increment HL reg pair by 1
	MOV A, M		Load second number in accumulator
	MVI C, (00)H		Initialize reg for carry; C= (00)H
	ADD B		[A] = [A]+ [B]
	JNC AHEAD		Jump; if carry = 0
	INR C		Increase reg C content by 1
AHEAD	INX H		
	MOV M, A		Store the Sum
	INX H		
	MOV M, C		Store the Carry
STOP	HALT		

Experiment 2: Write an ALP to multiply two 8-bit numbers that are stored in consecutive memory location $(****)_H$ & $(**** + 1)_H$ respectively. Store the result in next consecutive memory locations.

Label	Mnemonics	Hex Codes	Comments
START	LXI H, $(****)_H$		Load HL memory location of first number
	MOV A, M		Load first number in accumulator
	MOC C, A		Set counter for multiplication
	INX H		HL reg pair increase by 1
	MVI A, (00)H		Initialize accumulator by zero
BACK	ADD M		Addition
	DCR C		Decrement counter by 1
	JNZ BACK		Jump back if zero flag is not set
	INX H		
STOP	MOVE M, A		Result at respective memory location
STOP	HALT		

Experiment 3: Write an ALP to divide two 8-bit numbers that are stored in consecutive memory location $(****)_H$ & $(**** + 1)_H$ respectively. Store the quotient & reminder in $(YYYY)_H$ and $(YYYY + 1)_H$ memory locations respectively

Label	Mnemonics	Hex Codes	Comments
START	LXI H, (****)		Load memory address of dividend
	MOV A, M		Move dividend to accumulator
	MVI D, (00)H		Load D reg. by (00)H
	INX H		Increase HL reg pair
	CMP M		Compare memory data with dividend
	JC AHEAD		Jump ahead if carry is set
BACK	SUB M		Subtract dividend minus divisor
	INR D		Increase reg D
	CMP M		Compare memory
	JNC BACK		Jump back if carry is reset
	STA (****) $_H$		Store reminder
	MOV A, D		Store quotient
	STA (****) $_H$		
AHEAD	RST 1		Stop

Experiment 4: Write an ALP to transfer content of source memory area to the destination memory area.

Label	Mnemonics	Hex Codes	Comments
Start	LDA (****) H		Set counter
	MOV H, A		
	LXI B, (****) H		Load source address
	LXI D (****) H		Load destination address
BACK	LDAX B		Copy data to accumulator
	STAX D		Store data to destination
	INX B		Increase source address
	INX D		Increase destination address
	DCR H		Decrement counter
	JNZ BACK		Jump back if data is not transferred
Stop	RST 1		Stop

Experiment 5: Write an ALP to find out minimum number from an array of 3, 8-bit numbers.

Label	Mnemonics	Hex Codes	Comments
START	LDA (ZZZZ)H		Set counter for no. has to be verified
	MOC C, A		
	DCR C		Decrement reg c by 1
	LXI H (XXXX)H		Set memory pointer
	MOV A, M		Transfer the first number to Acc
BACK	INX H		Increment memory location by 1
	CMP M		Compare the [ML] with [A]
	JC (AHEAD)		Jump AHEAD if [A] < [ML] data
	MOV A, M		Copy the data from ML to ACC
AHEAD	DCR C		Decrement counter by 1
	JNZ (BACK)		Jump back if zero flag is not set
	STA (ZZZZ)H		Store the result
STOP	RST 1		Stop

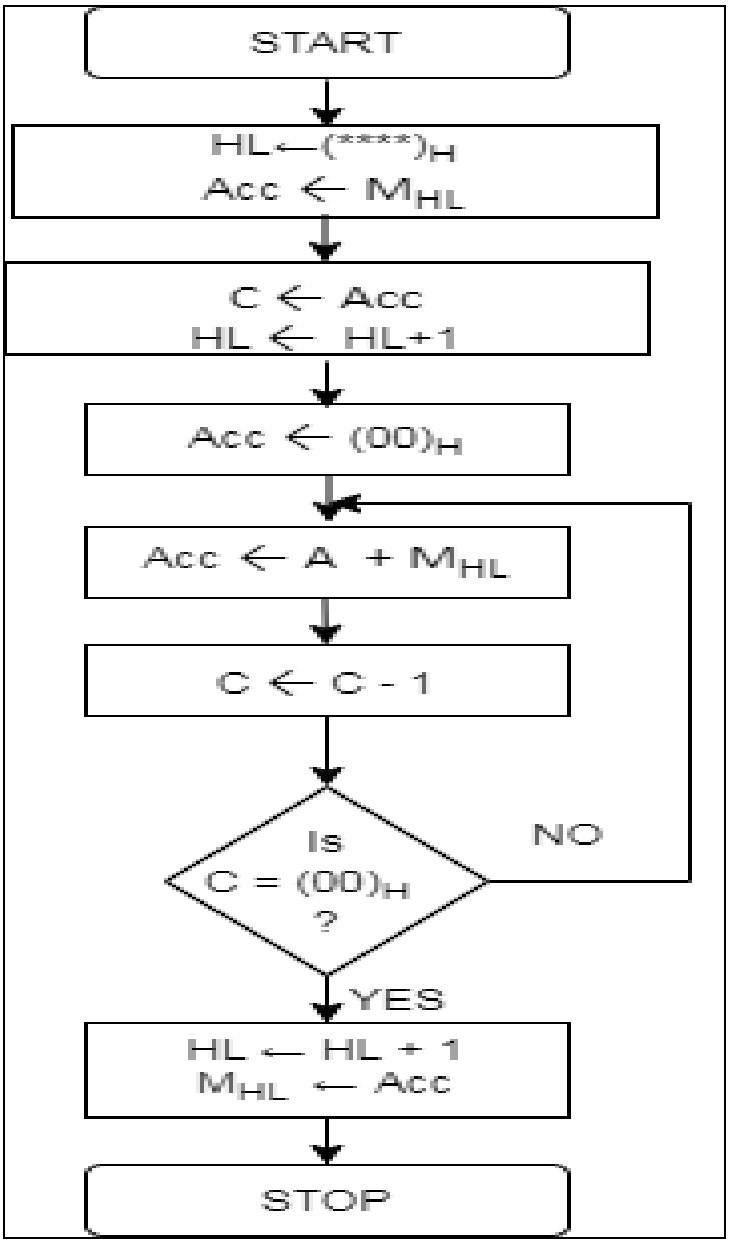
Experiment 6: Write an ALP to find even numbers from given array of 10, 8-bit numbers.

Label	Mnemonics	Hex Codes	Comments
START	MVI C, (0A) H		Set register C as counter
	MVI B, (00) H		Initialize reg B with zero
	LXI H (XXXX) H		Load HL with source ML for number
	LXI D (YYYY)H		Load DE with destination ML of ENs
BACK	MOV A, M		Copy the number from ML to ACC
	RRC		Rotate right the data of accumulator
	JC (AHEAD)		Jump AHEAD if carry is set
	INR B		Increment reg B by1
	MOV A, M		Copy the number from ML to ACC
	STAX D		Store even number at destination ML
	INX D		Increment DE reg pair
AHEAD	INX H		Increment HL reg pair
	DCR C		Decrement reg C by 1
	JNZ (BACK)		Jump BACK if zero flag is not set
	MOV A, B		Load total number of even numbers
	STA (ZZZZ)H		Store count of even numbers

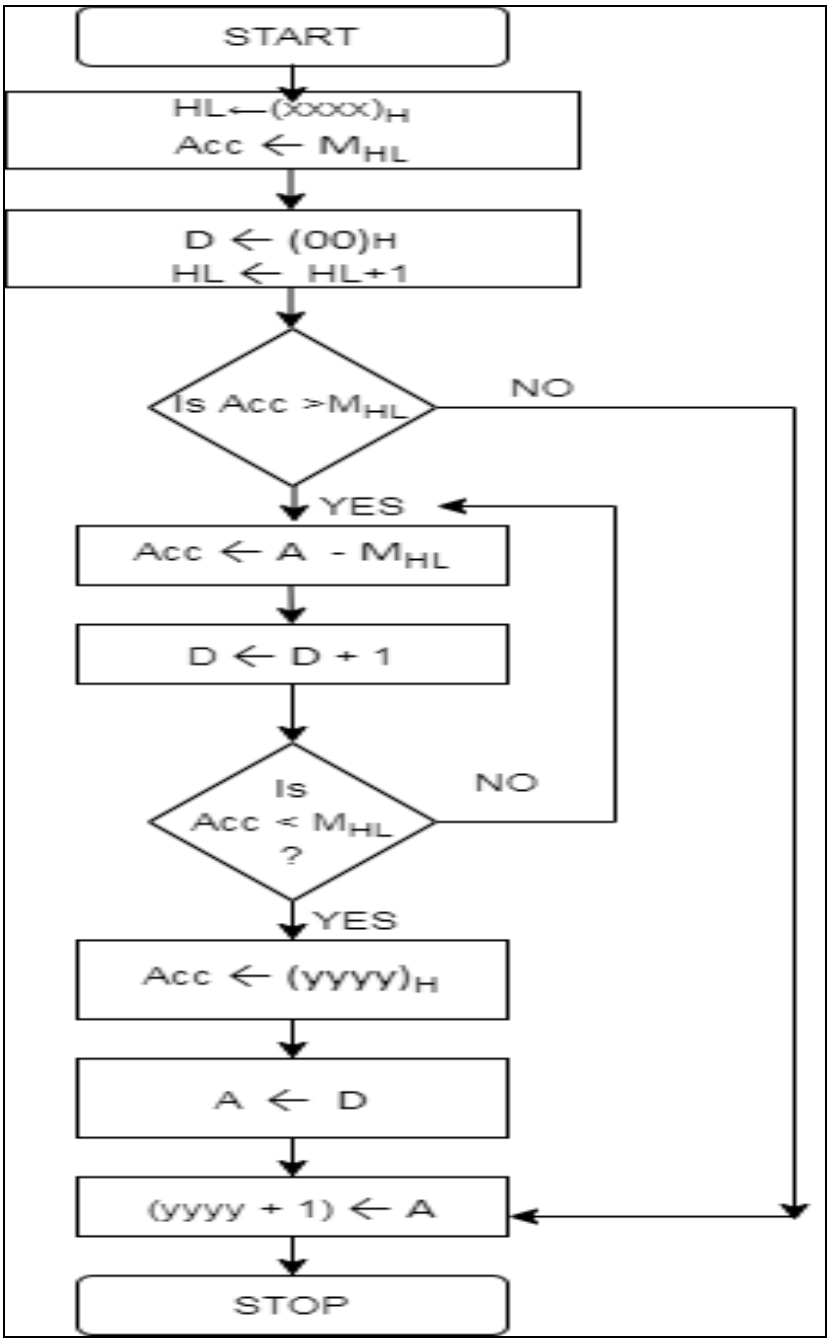
Flowcharts:

Below are the flowcharts of some of the above-mentioned programs. Debug, analyze and correlate with the respective programs mentioned above.

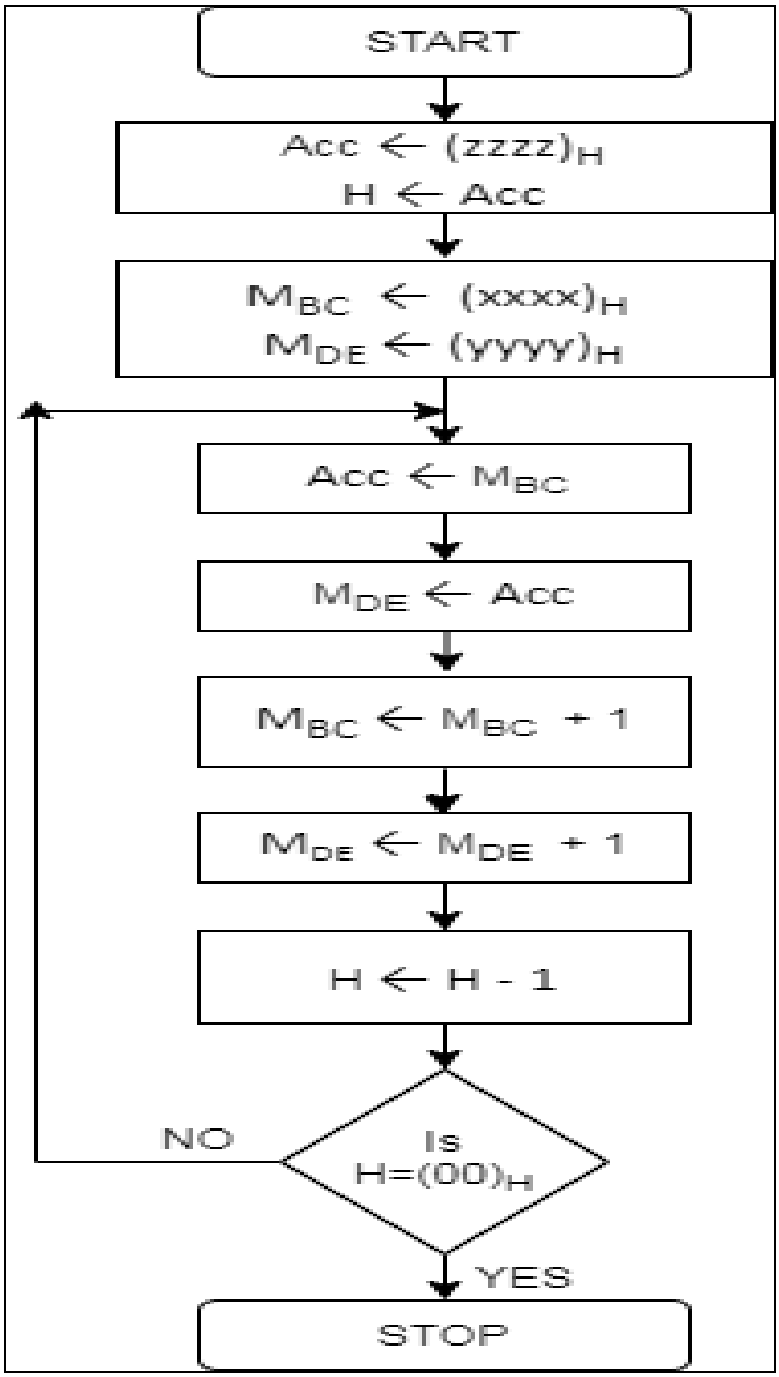
Flowchart 1:



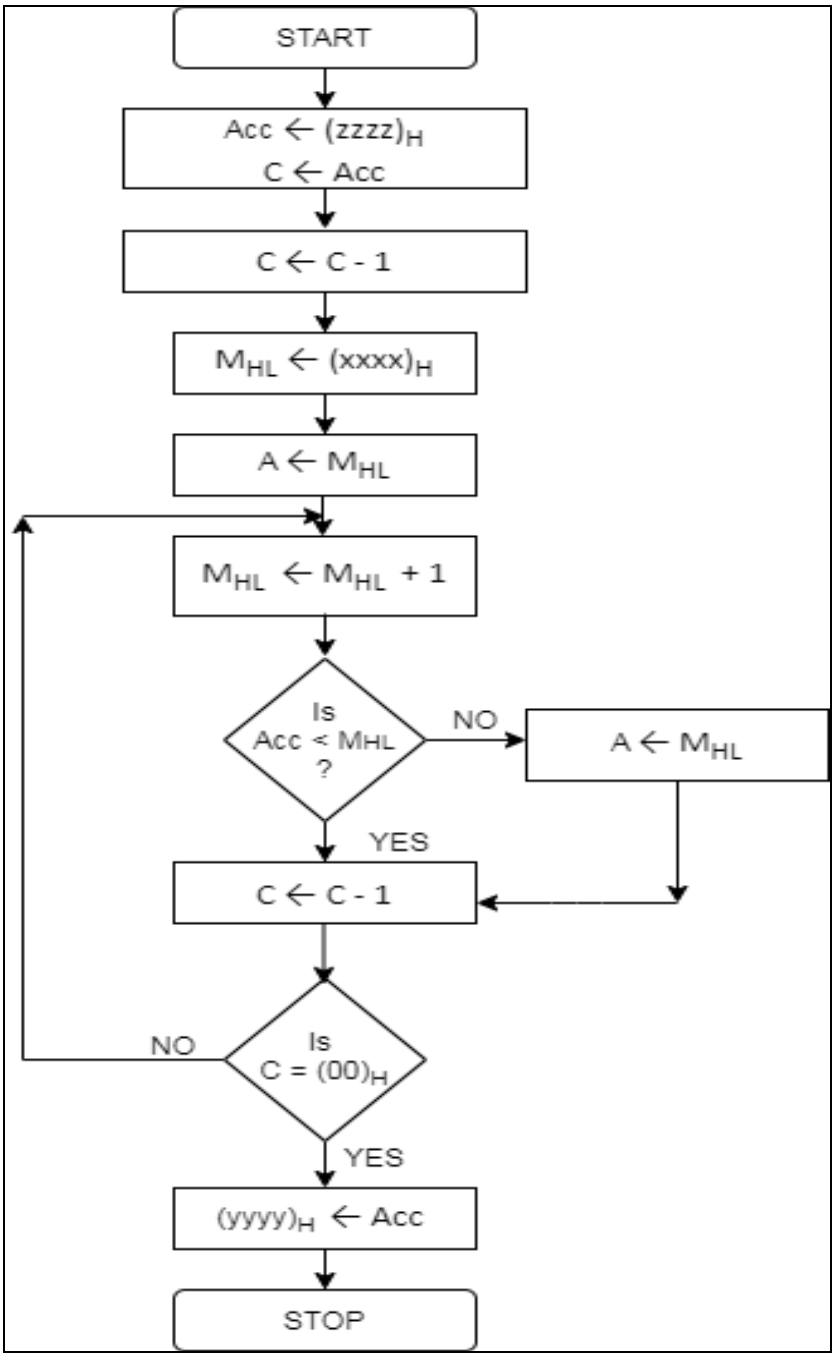
Flowchart 2:


START
HL←(XXXX)H
Acc ← MHL
D ← (00)H
HL ← HL+1
Is Acc >MHL
NO
YES
Acc ← A - MHL
D ← D + 1
Is
Acc < MHL
?
NO
YES
Acc ← (yyyy)H
A ← D
(yyyy + 1) ← A
STOP

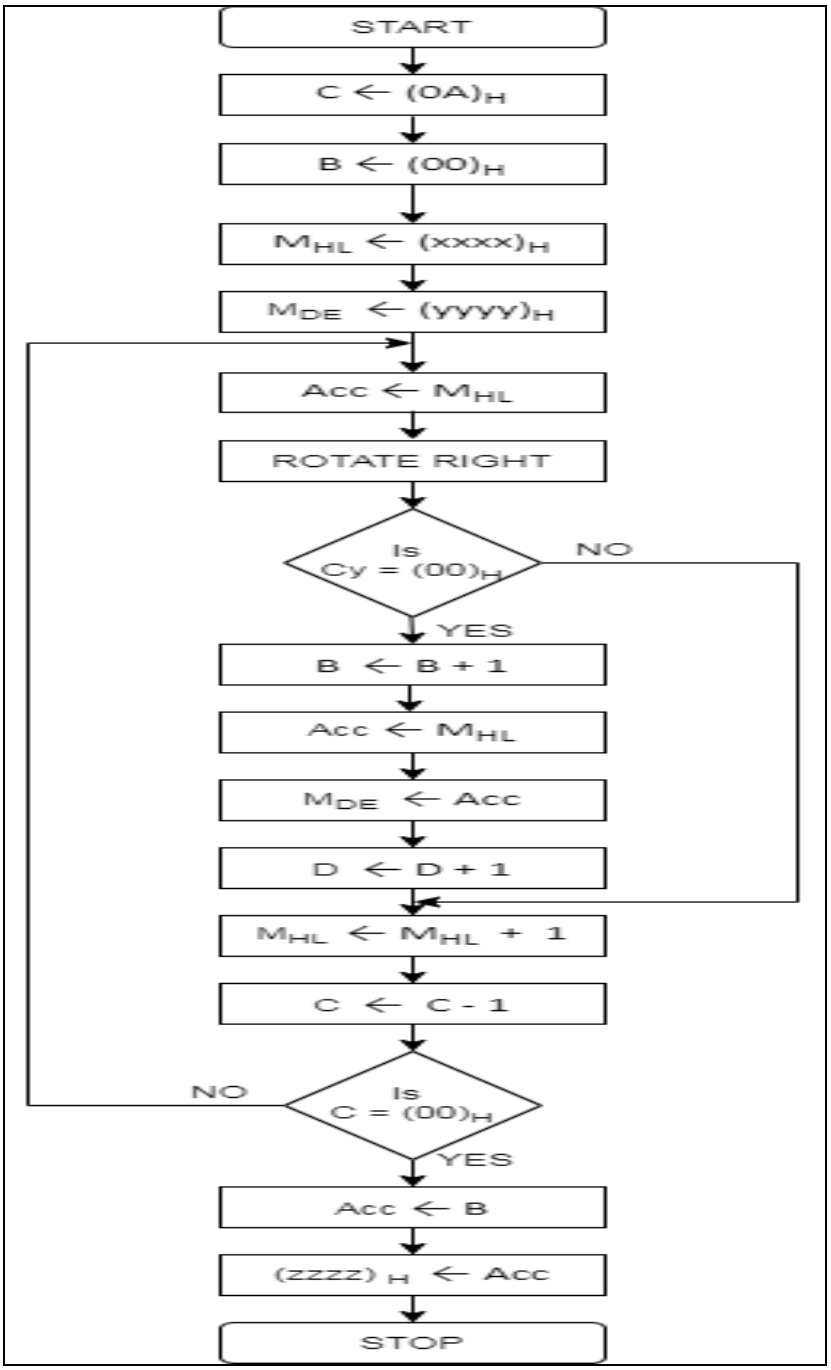
Flowchart 3:


START
Acc ← (zzzz)H
H ← Acc
MBC ← (xxxx)H
MDE ← (yyyy)H
Acc ← MBC
MDE ← Acc
MBC ← MBC + 1
MDE ← MDE + 1
H ← H - 1
NO
Is
H=(00)H
YES
STOP

Flowchart 4:


START
Acc ← (zzzz)H
C ← Acc
C ← C - 1
MHL ← (xxxx)H
A ← MHL
MHL ← MHL + 1
Is Acc < MHL ?
NO
A ← MHL
YES
C ← C - 1
Is C = (00)H
NO
YES
(yyyy)H ← Acc
STOP

Flowchart 5:


START
C ← (0A)H
B ← (00)H
MHL ← (xxxx)H
MDE ← (yyyy)H
Acc ← MHL
ROTATE RIGHT
Is Cy = (00)H
NO
YES
B ← B + 1
Acc ← MHL
MDE ← Acc
D ← D + 1
MHL ← MHL + 1
C ← C - 1
Is C = (00)H
NO
YES
Acc ← B
(zzzz) H ← Acc
STOP

Exercise:

1. Write an ALP to add two numbers stored in reg B and reg C respectively. Store the 16-bit result in the DE register pair.
2. Write an ALP to subtract two 8-bit numbers that are stored in two consecutive memory location (****) H and (**** + 1) H respectively. Select the numbers in such a way that the bigger number is to be subtracted from a smaller number. Justify the result.
3. Write an ALP to subtract two numbers stored in reg B from the number in reg C respectively. Store the result in E register pair.
4. Write an ALP to multiply two numbers stored in reg B and reg C respectively. Store the 16-bit result in the HL register pair.
5. Write an ALP to find out maximum number from an array of 3, 8-bit numbers stored from $(XXXX)_H$ and to store it at memory location at $(YYYY)_{H.}$ The number of elements in the series is stored at $(ZZZZ)_H$.
6. Write an assembly language program to find odd numbers from given array of ten 8-bit numbers stored from $(XXXX)_H$ onwards and to store them from $(YYYY)_H$ onwards & store number of odd numbers at $(ZZZZ)_H$.
7. Write an assembly language program to find positive numbers from a given array of ten 8-bit numbers stored from $(XXXX)_H$ onwards and to store them from $(YYYY)_H$ onwards & store number of positive numbers at $(ZZZZ)_H$.
8. Write an assembly language program to find negative numbers from given array of 10, 8-bit numbers stored from $(XXXX)_H$ onwards and to store them from $(YYYY)_H$ onwards & store number of negative numbers at (ZZZZ)

Chapter 13

OP-CODE SHEET: μP 8085 INSTRUCTIONS