

# FUNDAMENTALS OF MICROPROCESSOR



## UNIT 2

# PROGRAMMING THE 8085

(DATE: 16-09-20 TO 21-10-20)

By:

**ATIFA AQUEEL**

Guest Teacher

Electronics Engg. Section

University Women's Polytechnic, AMU

# SYLLABUS

## **I. INTRODUCTION TO THE INTEL 8085**

- Definition of microprocessor, generations and types of microprocessors, most popular microprocessors.
- Architecture of 8085, brief description of ALU, register section, data & address buses. Bus time sharing 8085 CPU pins and associated signal.

## **II. PROGRAMMING THE 8085**

- Instruction, Group of instruction, Addressing modes of Instruction, 8085 instruction set. Machine Language, Assembly Language comparison, Assembly Language programming (Simple Problems).

## **III. TIMING INSTRUCTION & EXECUTION**

- Machine, Instruction, Fetch, Read, Write(IO/MEM) cycle timing diagram, Interruption: Types of 8085 interrupt system, 8085 SID and SOD lines.

# SYLLABUS

## IV. PERIPHERAL INTERFACING

- PPLD'S, Brief description of 8255, 8257-DMA Controller, Introduction to Intel's 8086 microprocessor, Popular applications of Microprocessor in industry.

### ***BOOKS RECOMMENDED:-***

1. *Microprocessor Architecture, Programming & Applications, by Goankar, 6th Edition 2013*
2. *Fundamentals of Microprocessor and Microcontrollers, by B.Ram, Dhanpat Rai Publications, 9<sup>th</sup> edition 2019.*
3. *Microprocessor and Microcomputers by Rafiquzzaman, Universal Book Stall (UBS Pub.) 1997*
4. *Introduction to Microprocessor, by Mathur, 3rd Edition 1997, Aditya P Mathur (Tata McGraw Hill, Pub. Co.Ltd)*

# INSTRUCTION

An instruction is a command given to the computer to perform a specified operation on given data. The instruction set of a microprocessor is the collection of the instructions that the microprocessor is designed to execute. The instructions described in this chapter are of INTEL 8085. These instructions are of Intel Corporation. They cannot be used by other microprocessor manufacturers. The programmer can write a program in assembly language using these instructions. These instructions have been classified into the following groups:

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

# Opcodes and Operands

Each instruction contains two parts: operation code (opcode) and operand. The first part of an instruction which specifies the task to be performed by the computer is called *opcode*. The second part of the instruction is the data to be operated on, and it is called *operand*. The operand (or data) given in the instruction may be in various forms such as 8-bit or 16-bit data, 8-bit or 16-bit address, internal registers or a register or memory location. In some instructions the operand is implicit. When operand is a register it is understood that the data is the content of the register.

# Data Transfer Group

**Data Transfer Group.** Instructions which are used to transfer data from one register to another register, from memory to register or register to memory, come under this group. Examples are: MOV, MVI, LXI, LDA, STA etc. When an instruction of data transfer group is executed, data is transferred from the source to the destination without altering the contents of the source. For example, when MOV A, B is executed the content of the register B is copied into the register A, and the content of register B remains unaltered. Similarly, when LDA 2500 is executed the content of the memory location 2500 is loaded into the accumulator. But the content of the memory location 2500 remains unaltered.

- **For e.g.**
  1. MOV A,B : Move the content of reg B to reg A
  2. MVI A, 25H : Move 25H to accumulator (Reg A)
  3. LDA 2000H : Load the accumulator with the content present in memory location 2000H.
  4. LXI B, 2020H: Load the register pair BC with the content present in memory location 2020H and 2021H. (X: Reg pair ...16 bits)

# Arithmetic Group

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

- **For e.g.**
  1. **ADD B:**
  2. **SUB C:**
  3. **INR A:**
  4. **DCR C:**

# Other 3 Groups

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

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

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



# Instruction Word Size

## 3.1.8 Instruction Word Size

A digital computer understands instructions written in binary codes (machine codes). The machine codes of all instructions are not of the same length. According to the word size the Intel 8085 instructions are classified into the following three types:

- (1) 1-byte instruction
- (2) 2-byte instruction
- (3) 3-byte instruction

# One Byte Instruction

- The instruction which has only one part i.e. opcodes is known as one byte instruction. In other words one byte instructions include the opcode and operand in the same byte and it is stored in only one memory location.

**One-Byte instruction.** Examples of one-byte instructions are:

**MOV A, B.** Move the content of register B to register A. 78H is the opcode for MOV A, B. Besides the operation to be performed the opcode also specifies the registers which contain operands (data). The opcode 78H can be written in the binary form as 01111000. The 1st two bits, i.e., 01 are for MOV operation; the next three bits 111 are the binary code for register A and the last three bits 000 are the binary code for register B.

**ADD B.** Add the content of register B to the content of the accumulator. 80H is the opcode for the instruction ADD B. In this instruction one of the operands is register B (its content) which is indicated in the instruction itself. In this type of instruction (arithmetic group of instruction) it is assumed that the other operand is in the accumulator. The opcode 80H in the binary form is 10000000. The first five bits, i.e. 10000 specify the operation to be performed, i.e. ADD operation. The last three bits 000 are the code for register B for 8085 microprocessor.

**RAL.** Rotate the content of the accumulator left by one bit. The opcode for RAL is 17H. In this instruction operand is not given, it is implied. The operand is in the accumulator. The instruction has to operate on the content of the accumulator.

All the above three examples are only one byte long. All one-byte instructions contain information regarding operands in the opcode itself.

# Two-Byte Instruction

- In two byte instruction , the first byte is in its operand and the second byte is either 8-bit data or address. For Example:

opcode and the 2nd byte is either data or address.

(i) MVI B, 05 ; Move 05 to register B.

06, 05 ; MVI B, 05 in the code form.

The 1st byte 06 is the opcode for MVI B and the 2nd byte 05 is the data which is to be moved to register B.

(ii) IN 01 ; Read data at port B.

DB, 01 ; IN 01 in the code form.

DB is the opcode for the instruction IN and 01 is the address of a port (the port B of Intel 8255.1 of a microprocessor kit). A two-byte instruction is stored in two consecutive memory locations.

# Three Byte Instruction

**Three-Byte Instruction.** In a three byte instruction the 1st byte of the instruction is its opcode and the 2nd and 3rd bytes are either 16-bit data or 16-bit address. Examples are:

- (i) LXI H, 2400H ; Load H-L pair with 2400H.  
21, 00, 24 ; LXI H, 2400H in the code form.

The 1st byte 21 is the opcode for the instruction LXI H. The 2nd byte 00 is 8 LSBs of the data (2400H), which is loaded into register L. The 3rd byte 24 is 8 MSBs of the data (2400H), which is loaded into register H.

- (ii) LDA 2500H ; get the content of the memory location 2500H into accumulator.

3A, 00, 25 ; LDA 2500H in the code form.

The 1st byte 3A is the opcode for the instruction LDA. The 2nd byte 00 is 8 LSBs of the address of the memory location 2500H. The 3rd byte 25 is 8 MSBs of the address of the memory location 2500H. In this instruction the data is the content of the memory location 2500H. The data is to be loaded into the accumulator. A 3-byte instruction is stored in three consecutive memory locations.

# Q: Write the size of the following instructions

1. SUB B : 1 BYTE
2. INR B : 1 BYTE
3. MVI B,25H : 2 BYTE
4. LXI D, 2500H : 3 BYTE
5. MOV M, A : 1 BYTE
6. SBI 38H : 2 BYTE
7. INX H : 1 BYTE
8. DCR D : 1 BYTE
9. CMA : 1 BYTE
10. RAL : 1 BYTE
11. LHLD 3000H : 3 BYTE

# Instruction and Data Formats

- The various techniques to specify data for instructions are:
  1. 8-bit or 16-bit data may be directly given in the instruction itself.
  2. The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
  3. In some instructions, only one register is specified. The content of the specified register is one of the operands.
  4. Some instructions specify two registers. The contents of the registers are the required data.
  5. In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.
- Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length. It may 1-byte, 2-byte or 3-byte instruction.

# 8085 Addressing Modes

- Each instruction requires certain data on which it has to operate.
- The different technique by which the microprocessor specifies the operand in an instruction is known as addressing mode.
- Intel 8085 supports the following addressing modes:
  1. Immediate Addressing Modes
  2. Register Addressing Modes
  3. Direct Addressing Modes
  4. Indirect Addressing Modes
  5. Implicit Addressing Modes

# Immediate Addressing Mode

- In this mode, the 8/16-bit operand (data/address) is specified in the instruction itself as one of its operand.
- In this mode, the number directly given by the programmer is moved into the location specified. This location could be a register or an address in the memory.
- In other words, we can understand the Immediate Addressing Mode as **a mode in which an immediate value is given to operate on.**
- For Example:
  1. MVI A, 25H ; Save 25H into the Accumulator
  2. MVI C, 62H ; Save 62H into the C register
  3. LXI B, 3050H ;Load B-C pair with value 3050H.
  4. ADI 20H ;The value 20H is added to the contents of the accumulator.
  5. SUI 22H ;The value 22H is subtracted from the contents of the accumulator



# Immediate Addressing Mode

- Some other examples are:
  1. **LXI H, 3050H** ;Load H-L pair with value 3050H. Higher byte 30H goes in H while lower byte 50H goes in L
  2. **LXI SP, 4050H** ;Load value 4050H in stack pointer register
  3. **JMP 9000H** ;Jump to address 9000H.
- **Note:** These instructions may seem to be too many. But you can simply remember them as “**The instructions ending with the letter I, which stands for immediate, are the instructions for immediate addressing mode**” with one exception though: **the JMP** instruction.

# Register Addressing Mode

- In register addressing mode the operand is in one of the general purpose registers.
- The opcode specifies the address of the register in addition to the operation to be performed.
- In this mode, the data to be accessed and operated upon is present in a register and is accessed by specifying the name of the register.
- For Example:
  1. `MOV A, B` ;Save contents of B in Accumulator
- This command moves the contents stored inside the B register to the Accumulator.
- 2. `ADD B` ;Adds the value stored in B to the value in Accum.
- 3. `SUB E` ;Subtracts the value stored in E to that in Accum.
- 4. `INR C` ;Increments the contents of register C by 1
- 5. `INX B` ;Increments the contents of register pair BC by 1

# Direct Addressing Mode

- In this mode, the address of the operand is specified in the instruction itself.
- For example LDA 3000H: means the data at address 3000H is copied to register A.
- In other words, we access the data stored in a memory location using its address. Consider the example
  1. **LDA 2034H** ;Load the content at memory location 2034H into the Accumulator
  2. **LHLD 2040H** ;Load the content at 2040H into register L and contents at the next location 2041H into register H
  3. **STA 3030H** ;Load the content at memory location 3030H into the Accumulator
  4. **SHLD 2040H** ;Load the contents of the register L into memory location with address 2040H and load the ;contents of register H at the next memory location 2041H

# Program 1

- Write a program to add two numbers present in memory locations 2000H and 2001H and add 25H to the result and store the final result in memory location 4000H.


## Memory

Address	Data
2000H	20H
2001H	30H
...	
...	
4000H	.....

# Solution

- LDA 2000H : Load Acc with the content of mem location 2000H (A = 20H)
- MOV B,A (B=A=20H)
- LDA 2001H (A=30H)
- ADD B (A+B) AND RESULT WILL BE STORED IN ACC
- ADI, 25H (A+25)
- STA 4000H ( 4000H= A)
- HLT

# Register Indirect Addressing Mode

- In this mode, the register(s) holds the address of the location from which data is to be retrieved.
- Here the address of the operand is specified in a register pair, Hence this type of instructions indirectly point to the operand.
- MOV A, M ;move contents of the memory location whose address is held by HL pair into the Accumulator. 

(1) LXI H, 2500 H    Load H-L pair with 2500 H.

MOV A, M            Move the content of the memory location, whose address is in H-L pair (*i.e.* 2500 H) to the accumulator

HLT                  Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, *i.e.* LXI H, 2500 H.

# Example

- Consider the following lines of instructions.
- Suppose it is known that the number 25H is stored at memory location with address 2030H and the programmer wants to transfer that data to accumulator.
- MVI H, 20H }  
• MVI L, 30H } OR LXI H, 2030H
- MOV A, M ;M stands for memory (i.e HL pair)
- MVI M, 92H
- After running the above code, register A will hold the value 25H.

# Explanation

1. After the execution of the first line, register H will have value 20H stored in it.
2. After the execution of the second line, register L will have value 30H stored in it. So, now we have 2030H stored in register pair HL.
3. The instruction in the third line tells the microprocessor to look at the address (in the memory) given by the HL pair which is now 2030H. So, the microprocessor looks at the location with address 2030H and loads the content stored there into the accumulator. As we assumed, the memory location with address 2030H had the number 25H stored in it. So, the microprocessor will copy it and paste it in the accumulator. And hence, the accumulator will contain 25H.
4. In the fourth line of instruction, we tell the microprocessor to move the value given by us, i.e. 92H to the memory location. And the address of the memory location is read from the HL register pair. HL pair still holds the value 2030H. So, the microprocessor will store 92H at the memory location with address 2030H.
5. So, in the end, the Accumulator will hold the value 25H and the memory location with address 2030H will hold the value 92H.



# Register Indirect Addressing Mode

- A **very important point** to note here is that the address of a memory location is of 16-bit size but the data stored in a single memory location is 8 bits.
- Some more instructions which use this mode of addressing.
  1. ADD M ;Add contents of the memory location whose address is held by HL pair to the contents of the Accumulator
  2. INR M ;Add contents of the memory location whose address is held by HL pair is incremented by 1
  3. CMP M ;16 bit number from HL pair is picked up. Then, contents at the address given by picked up number is compared with those of the accumulator.
- The result of the comparison is shown by setting the flags of the PSW as follows: ;
  - \* if  $(A) < (\text{reg/mem})$ : carry flag is set ;
  - \* if  $(A) = (\text{reg/mem})$ : zero flag is set ;
  - \* if  $(A) > (\text{reg/mem})$ : carry and zero flags are reset

# Implied Addressing Mode

- In this mode of addressing, the microprocessor already knows the location of data to process and the programmer does not need to define it explicitly. That information is already contained in the opcode.
- For example,
  1. CMA ; Each bit of the 8 bit number stored in Accumulator is complemented. Note that in this case, it is implied ; that the data to be processed is in the accumulator and we don't need to specify it.
  2. RLC ; the 8 bits of the number stored in the Accumulator are manipulated in such a way that each bit is moved ; left by 1 place and the most significant bit is moved to the least significant position.
  3. CMC ; The carry flag is complemented
  4. STA ; The carry flag is set to 1
- The address of data to be manipulated is the register A which is not stated in the instruction. It is already understood by the microprocessor when such instructions (instructions with implied addressing mode) are used.

# Symbols and Abbreviations

- The symbols and abbreviation used in INTEL 8085 programming are:

Symbol/Abbreviations	Meaning
Addr	16-bit address of the memory location.
Data	8-bit data
data 16	16-bit data
r, r1, r2	One of the registers A, B, C, D, E, H or L
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
PSW	Program Status Word

# Symbols and Abbreviations

M	Memory whose address is in H-L pair
H	Appearing at the end of the group of digits specifies hexadecimal, e.g. 2500H
Rp	One of the register pairs.
Rh	The high order register of a register pair
RI	The low order register of a register pair
PC	16 bit program counter, PCH is high order 8 bits and PCL low order 8 bits of register PC.
CS	Carry Status
[ ]	The contents of the register identified within bracket
[ [ ] ]	The content of the memory location whose address is in the register pair identified within brackets
$\wedge$	AND operation
$\vee$	OR operation
$\oplus$ or $\nabla$	Exclusive OR
$\leftarrow$	Move data in the direction of arrow
$\Leftrightarrow$ 21-10-2020	Exchange contents



# DATA TRANSFER GROUP

- Instructions which are used to transfer the data from a register to another register from memory to register or register to memory come under this group.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles	Example
MOV $r_1, r_2$ $[r1] \leftarrow [r2]$	Move the content of the one register to another	4	none	Register	1	MOV A, B
MOV $r, M$ $[r] \leftarrow [[H-L]]$	Move the content of memory to register	7	none	Register Indirect	2	MOV B, M
MOV $M, r$ $[[H-L]] \leftarrow [r]$	Move the content of register to memory	7	none	Register Indirect	2	MOV M, C
MVI $r, data$ $[r] \leftarrow data$	Move immediate data to register	7	None	Immediate Register	3	MVI M, 08
LXI $rp, data\ 16$ $[rp] \leftarrow data\ 16\ bits, [rh] \leftarrow 8\ MSBs,$ $[rl] \leftarrow 8\ LSBs\ of\ data$	Load Register pair immediate	10	None	Immediate	3	LXI H, 2500H

LDA addr [A] ←[addr]	Load Accumulator direct	13	None	Direct	4	LDA 2400 H
STA Addr [addr] ←[A]	Store accumulator direct	13	None	Direct	4	STA 2000H
LHLD addr [L] ←[addr], [H] ← [addr + 1 ]	Load H-L pair direct	16	None	Direct	5	LHLD 2500H
SHLD addr [addr] ←[L], [addr +1] ← [H]	Store H-L pair direct	16	None	Direct	5	SHLD 2500 H
LDAX rp [A] ←[[rp]]	Load accumulator indirect	7	None	Register Indirect	2	LDAX B
STAX rp [[rp]] ←[A]	Store accumulator indirect	7	None	Register Indirect	2	STAX D
XCHG [H-L] ↔[D-E]	Change the contents of H-L with D-E pair	4	None	Register	1	

# Examples

Q.1: Place 28H in register C.

Q.2: Place 05H in accumulator and transfer it in register C.

Q.3: Load the content of memory location 4050 directly to the accumulator and transfer it to register B. The content of memory location 4050H is 25H.

Q.4: Move the content of memory location 2010H in register B indirectly, the content of memory location 2010H is 1AH.

Q.5: Place 05H in accumulator. Increment it by one and store the result in memory location 2400H.

# Arithmetic Group

- The instructions of this group perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or a memory.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles	Example
ADD r $[A] \leftarrow [A] + [r]$	Add register to accumulator	4	All	Register	1	ADD B
ADD M $[A] \leftarrow [A] + [[H-L]]$	Add memory to accumulator	7	All	Register indirect	2	ADD M
ACC r $[A] \leftarrow [A] + [r] + [CS]$	Add register with carry to accumulator	4	All	Register	1	ADC B
ADC M $[A] \leftarrow [A] + [[H-L]] [CS]$	Add memory with carry to accumulator	7	All	Register indirect	2	ADC M
ADI data $[A] \leftarrow [A] + \text{data}$	Add immediate data to accumulator	7	All	Immediate	2	ADI 20H



ADI data [A] ← [A] + data	Add immediate data to accumulator	7	All	Immediate	2	ADI 20H
ACI data [A] ← [A] + data + [CS]	Add with carry immediate data to accumulator	7	All	Immediate	2	ACI 20H
DAD rp [H-L] ← [H-L] + [rp]	Add register pair to H-L pair	10	CS	Register	3	DAD B
SUB r [A] ← [A] - [r]	Subtract register from accumulator	4	All	Register	1	SUB B
SUB M [A] ← [A] - [[H-L]]	Subtract memory from accumulator	7	ALL	Register indirect	2	SUB M
SBB r/SBB M [A] ← [A] - [H-L] - [CS]	Subtract memory from accumulator with borrow	7	All	Register indirect	2	SBB B /SBB M
SUI data [A] ← [A] - data	Subtract immediate data from accumulator	7	All	Immediate	2	SUI 55H
SBI data [A] ← [A] - data - [CS]	Subtract immediate data from accumulator with borrow	7	All	Immediate	2	XCHG SBI 25H

INR r $[r] \leftarrow [r] + 1$	Increment register content	4	All except carry flag	Register	1	INR B
INR M $[[H-L]] \leftarrow [[H-L]] + 1$	Increment memory content	10	All except carry flag	Register indirect	3	INR M
DCR r $[r] \leftarrow [r] - 1$	Decrement register content	4	All except carry flag	Register	1	DCR B
DCR M $[[H-L]] \leftarrow [[H-L]] - 1$	Decrement memory content	10	All except carry flag	Register indirect	3	DCR M
INX rp $[rp] \leftarrow [rp] + 1$	Increment memory content	6	None	Register	1	INX B
DCX rp $[rp] \leftarrow [rp] - 1$	Decrement register pair	6	None	Register	1	DCX B
DAA	Decimal adjust accumulator	4			1	DAA

# DAA Instruction

- DAA: Decimal Adjust Accumulator
- The instruction DAA is used in the program after ADD, ADC, ADI, ACI etc.
- After ADD, ADC etc. the result is in hexadecimal form and it is placed in accumulator.
- The DAA instruction operates on this result and gives the final result in decimal system.
- It uses carry and auxiliary carry for decimal adjustment.
- When the sum lies between 10 to 15 (or A to F hexadecimal) a correction of +6 is made by DAA instruction. 6 is added to both LSBs and MSBs.
- 6 is added to 4 LSBs of the content of accumulator if the values lie in between A and F or the auxiliary carry flag is set to 1.
- Similarly, 6 is added to 4 MSBs of the content of accumulator if the values lie in between A and F or the carry flag is set to 1.
- Thus DAA instruction checks whether there is a carry, auxiliary flag or the sum lies between A to F and makes corrections accordingly.

# Example

- Let us consider we want to add two decimal numbers 38 and 45. They will be represented in BCD as 0011 1000 and 0100 0101. The addition results in 0111 1101. But the answer will be incorrect if we want to interpret this result as a BCD number. The result will be not only incorrect but also illegal as 1101, which we obtained as the last nibble in the answer is not a valid BCD number. Here, in such situations, we can use DAA to have the BCD sum as outcome. All that is required to be done is to add the BCD numbers and store the result in A, and then execute the DAA instruction.

Here is the detailing of the calculations –

```
38 ----> 0011 1000
+ 45 ----> 0100 0101
-----
83      0111 1101
-----
          7    D
```

- The illustration of the remedial actions against the previous example –

```

  38 ---> 0011 1000
+ 45 ---> 0100 0101
-----
  83      0111 1101      1
                        0111 1101
                        + 0110 (06H)
                        -----
                        1000 0011 ---> 83 (Decimal sum)

```

### Q.1: ADD 96D and 69D, 84D and 92D, 98D and 99D

- 96D: 1001 0110 (BCD)
- 69D: 0110 1001 (BCD)
- 165D: 1111 1111 (FF IN HEX)
- DAA : 0110 0110 (+6 correction to LSBs AND +6 correction to MSBs)
- 01,65: 1, 0110 0101 (BCD)

# Examples

**Q.1: Multiply 1245H by 2 .**

**Ans: LXI H, 1245H**

**DAD H**

**HLT**

**Q.2: ADD 1245H & 1345H using DAD instruction.**

**Ans: LXI H, 1245H**

**LXI B, 1345H**

**DAD B**

**HLT**

**Q.3: ADD two numbers present in memory locations 2501H and 2502H respectively and store the 8-bit result in 2503H (Use indirect addressing mode)**

# Indirect Addressing

## Program

```
LXI H, 2501H : "Get address of first number in H-L pair. Now H-L points to 2501H"
MOV A, M     : "Get first operand in accumulator"
INX H        : "Increment content of H-L pair. Now, H-L points 2502H"
ADD M        : "Add first and second operand"
INX H        : "H-L points 4002H"
MOV M, A     : "Store result at 2503H"
HLT          : "Stop"
```

## Example

```
(2501 H) = 99H
(2502 H) = 39H
Result (2503 H) = 99H + 39H = D2H
Since,
  1 0 0 1 1 0 0 1 (99H)
+ 0 0 1 1 1 0 0 1 (39H)
  1 1 0 1 0 0 1 0 (D2H)
```

# Logical Group

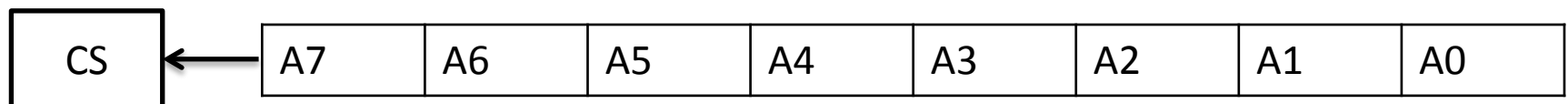
- The instructions in this group perform logical operation such as AND, OR, compare, rotate, etc.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
ANA r [A] ← [A] ∧ [r]	AND register with accumulator	4	All	Register	1
ANA M [A] ← [A] ∧ [[H-]]	AND memory with accumulator	4	All	Register indirect	2
ANI data [A] ← [A] ∧ [data]	AND immediate data with accumulator	7	All	Immediate	2
ORA r [A] ← [A] ∨ [r]	OR-register with accumulator	4	All	Register	1
ORA M [A] ← [A] ∨ [[H-L]]	OR-memory with accumulator	7	All	Register indirect	2
ORI data [A] ← [A] ∨ [data]	OR -immediate data with accumulator By: Atifa Aqueel	7	All	Immediate	2

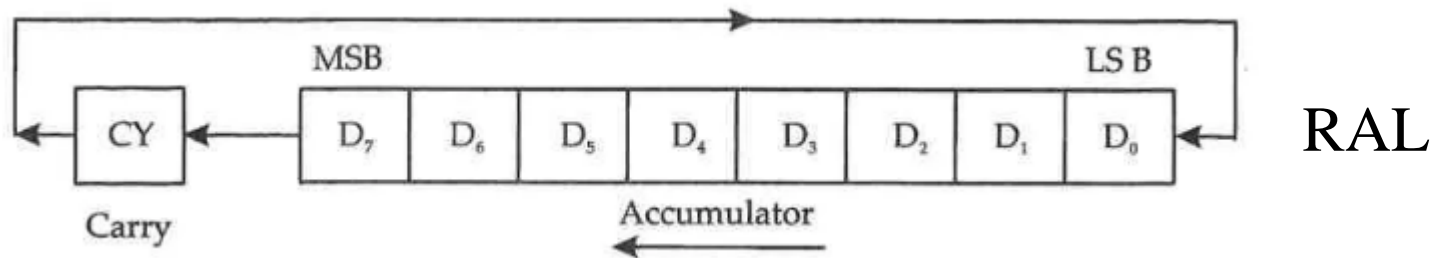
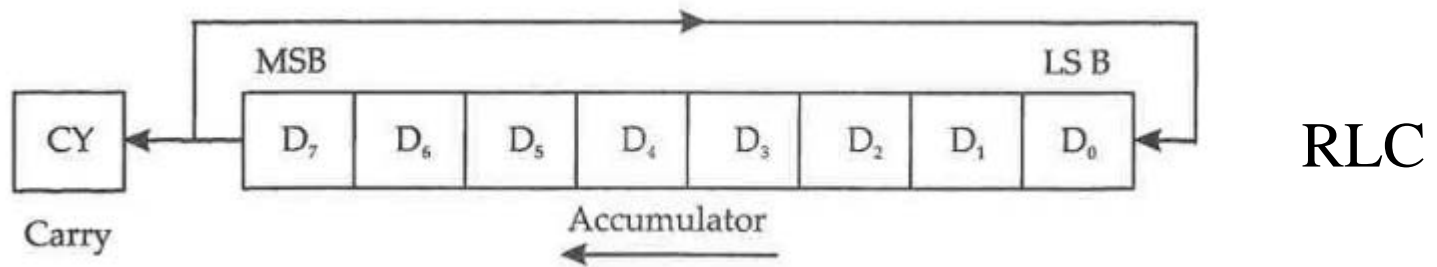


XRA r [A] $\leftarrow [A] \vee [r]$	XOR register with accumulator	4	All	Register	1
XRA M [A] $\leftarrow [A] \vee [[H-L]]$	XOR memory with accumulator	7	All	Register indirect	2
XRI data [A] $\leftarrow [A] \vee [data]$	XOR immediate data with accumulator	7	All	Immediate	2
CMA [A] $\leftarrow [A]$	Complement the accumulator	4	None	Implicit	1
CMC [CS] $\leftarrow [CS]$	Complement the carry status	4	CS		1
STC [CS] $\leftarrow 1$	Set carry status	4	CS		1
CMP r [A] - [r]	Compare register with accumulator	4	All	Register	1
CMP M [A] - [[H-L]]	Compare memory with accumulator	7	All	Register indirect	2
CPI data [A] - data	Compare immediate data with accumulator	7	All	Immediate	2

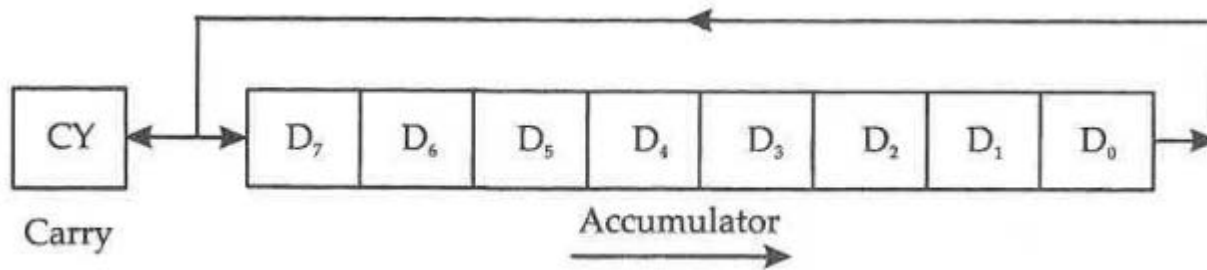
RLC $[A_{n+1}] \leftarrow [A^n], [A^0] \leftarrow [A^7], [CS] \leftarrow [A^7]$	Rotate accumulator left	4	Cs	Implicit	1
RRC $[A^7] \leftarrow [A^0], [CS] \leftarrow [A^0], [A^n] \leftarrow [A^{n+1}]$	Rotate accumulator right		CS	Implicit	1
RAL $[A^{n+1}] \leftarrow [A^n], [CS] \leftarrow [A^7], [A^0] \leftarrow [CS]$	Rotate accumulator left through carry		CS	Implicit	1
RAR $[A^n] \leftarrow [A^{n+1}], [CS] \leftarrow [A^0], [A^7] \leftarrow [CS]$	Rotate accumulator right through carry		CS	Implicit	1



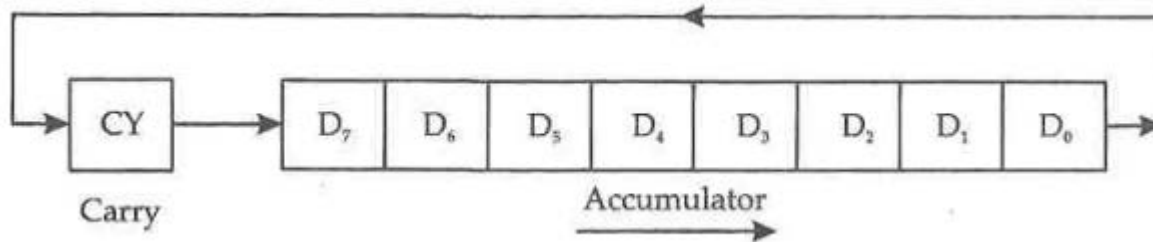
# RLC AND RAL



# RRC AND RAR



RRC



RAR

# Application of Rotate Instructions

- The rotate instructions can also be used for quick multiplication and division. The 8085 does not have special instructions for multiply and divide.
- Take a binary number. 0100. That's 4 in decimal. Shift the '1' towards left by one position. You get 1000. That's 8. There you go, you just multiplied it by 2. Shift the '1' in 0100 towards the right, and you get 2.
- To check whether the number is positive or negative.
- To check whether the number is even or odd.
- To find the number of 1's and 0's in a given number.

# PROGRAMS

1. WAP to find one's complement of an 8-bit number present in memory location 2501H and store the result in 2502H.
2. WAP to mask off the least significant 4 bits of an 8-bit number. Assume the no is present in memory location 2504H.
3. WAP to set the lower nibble without affecting the higher nibble.
4. WAP to complement the higher nibble without affecting the lower nibble.

# Solutions

1. LDA 2501H

- CMA
- STA 2502H
- HLT

2. LDA 2504H

- ANI 0FH
- STA 2505H
- HLT

3. LDA 2504H

- ORI 0FH
- STA 2505H
- HLT

4. LDA 2504H

- XRI F0H
- STA 2505H
- HLT

# Branch and Control Group

- The instructions of this group change the normal sequence of programs.
- There are two types of branch instructions:
- Conditional and Unconditional branch.
- The conditional branch instructions transfer the program to the specified label or memory location when certain conditions are satisfied.
- The unconditional branch instructions transfer the program unconditionally to the specified label or memory location.



# Unconditional Jump

## Unconditional Jump

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
JMP addr(label) [PC] ← Label	Unconditional jump: jump to the instruction specified by the address	10	None	Immediate	3

- In this instruction, the second and third byte instruction byte give the address of the label where the program jumps.
- Address of the label is the address of the next instruction to be executed. The program jumps to that location unconditionally.

# Conditional Jump

- After the execution of conditional jump instruction the program jumps to the instruction specified by the addresses or label if the specified conditions are true.
- The program proceeds further in normal sequence if the specified condition is not fulfilled.

## Conditional Jump

Instruction Set	Explanation	States	Machine Cycles
Jump addr (label) [PC] ← Label	Conditional jump: jump to the instruction specified by the address if the specified condition is fulfilled	10, if true and 7, if not true	3, if true and 2, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
JZ addr (label) [PC] ← address (label)	Jump, if the result is zero	Jump if Z=1	7/10	None	Immediate	2/3
JNZ addr (label) [PC] ← address (label)	Jump if the result is not zero	Jump if Z=0	7/10	None	Immediate	2/3
JC addr (label) [PC] ← address (label)	Jump if there is a carry	Jump if CS =1	7/10	None	Immediate	2/3
JNC addr (label) [PC] ← address (label)	Jump if there is no carry	Jump if CS =0	7/10	None	Immediate	2/3
JP addr (label) [PC] ← address (label)	Jump if result is plus	Jump if S=0	7/10	None	Immediate	2/3
JM addr (label) [PC] ← address (label)	Jump if result is minus	Jump if S=1	7/10	None	Immediate	2/3
JPE addr (label) [PC] ← address (label)	Jump if even parity	The parity status P =1	7/10	None	Immediate	2/3
JPO addr (label) [PC] ← address (label)	Jump if odd parity	The parity status P =0	7/10	None	Immediate	2/3

# Program 5

**Q:** Write an assembly language program to perform addition of two hexadecimal numbers (99H and 98H) by **immediate addressing mode**. Store the result and carry in memory locations 2503H and 2504H respectively.

Memory Address	Label	Mnemonics/ Opcodes	Comments
2000H		<b>MVI C, 00H</b>	Clear Register C
2002H		<b>MVI A, 99H</b>	Move 99H in Accumulator
2004H		<b>MVI B, 98H</b>	Move 98H in Register B
2006H		<b>ADD B</b>	Add the content of A and B and store the result in A
2007H		<b>JNC AHEAD</b>	Jump to AHEAD if CY = 0
200AH		<b>INRC</b>	If CY=1, increment Reg C by 1, C=1
200BH	<b>AHEAD:</b>	<b>STA 2503H</b>	Store the result (Accum) in 2503H
200EH		<b>MOV A,C</b>	Move content of C (i.e. Carry) in Accum.
200FH		<b>STA 2504H</b>	Store the Carry (Accum) in 2504H
2012H		<b>HLT</b>	End the program

Q: Write an assembly language programs to perform addition of two hexadecimal numbers stored at memory location 2501H and 2502H by **direct addressing mode**. Store the result and carry in memory locations 2503H and 2504H respectively.

Memory Address	Label	Mnemonics/ OpCodes	Comments
2000		<b>LDA 2500H</b>	Load the content of 2500H (i.e. 00H) to Accumulator
2003		<b>MOV C,A</b>	Move Accumulator content (i.e. 00H) to reg C
2004		<b>LDA 2501H</b>	Load the content of 2501H (i.e. one operand) to Accumulator
2007		<b>MOV B,A</b>	Move Accumulator content (i.e. one operand) to reg B
2008		<b>LDA 2502H</b>	Load the content of 2502H (i.e. other operand) to Accumulator
200B		<b>ADD B</b>	Add the content of A and B and store the result in A
200C		<b>JNC AHEAD</b>	Jump to AHEAD if CY = 0
200F		<b>INRC</b>	If CY=1, increment Reg C by 1, C=1
2010	<b>AHEAD:</b>	<b>STA 2503H</b>	Store the result (Accum) in 2503H
2013		<b>MOV A,C</b>	Move content of C (i.e. Carry) in Accum.
2014		<b>STA 2504H</b>	Store the Carry (Accum) in 2504H
2017		<b>HLT</b>	End the program

# Addition (Indirect Addressing)

Memory Address	Label	Mnemonics/ Opcodes	Comments
2000		<b>LXI H, 2500H</b>	Load the value 2500H in HL pair (HL =2500H)
2003		<b>MOV C,M</b>	Move the content of memory location whose address is in HL pair to register C. (C=00H)
2004		<b>INX H</b>	Increment HL pair by 1 (i.e, HL=2501H)
2005		<b>MOV A,M</b>	The value of 2501H will be moved to Accumulator
2006		<b>INX H</b>	Increment HL pair by 1 (i.e, HL=2502H)
2007		<b>ADD M</b>	[A] ← [A] + [[H-L]]
2008		<b>JNC AHEAD</b>	Jump to AHEAD if CY = 0
200B		<b>INR C</b>	If CY=1, increment Reg C by 1, C=1
200C	<b>AHEAD:</b>	<b>INX H</b>	Increment HL pair by 1 (i.e, HL=2503H)
200D		<b>MOV M,A</b>	Move the value of Acc (Result) to memory location 2503H
200E		<b>INX H</b>	Increment HL pair by 1 (i.e, HL=2504H)
200F		<b>MOV M, C</b>	Move the value of Reg C(carry) to memory location 2504H
2010		<b>HLT</b>	Stop the program

# Program 6

Q. WAP to Add the 16-bit number in memory locations 2501H and 2502H to the 16-bit number in memory locations 2503H and 2504H. Store the result in memory locations 2505H and 2506H with the most significant byte in memory location 2506H.

## Example

(2501H) = 15H

(2502H) = 1CH

(2503H) = B7H

(2504H) = 5AH

Result = 1C15 + 5AB7H = 76CCH

(2505H) = CCH

(2506H) = 76H

# Solution (without carry)

**Add two 16-bits numbers with DAD instruction**

```
LHLD 2501H : "Get 1st 16-bit number"
XCHG      : "Save 1st 16-bit number in DE"
LHLD 2503H : "Get 2nd 16-bit number in H-L"
DAD D     : "Add DE and HL"
SHLD 2505H : "Store 16-bit result in memory locations 2505H and 2506H".
HLT       : "Stop"
```



# Solution (with carry)

Memory Address	Label	Mnemonics/ Opcodes	Comments
2000H		<b>XRA A</b>	Clear the accumulator by xoring its value. A=00H
2001H		<b>LHLD 2501H</b>	Load HL pair with the value of 2501H AND 2502h
2004		<b>XCHG</b>	Exchange HL pair with DE pair
2005		<b>LHLD 2503H</b>	Load HL pair with the value of 2503H AND 2504h
2008		<b>DAD D</b>	Add HL and DE pair and store the result in HL pair
2009		<b>JNC AHEAD</b>	Jump to AHEAD if CY = 0
200C		<b>INR A</b>	If CY=1, increment Reg A by 1, A=1
200D	<b>AHEAD:</b>	<b>SHLD 2505H</b>	Store the value of HL pair (Result) in 2505H and 2506H.
2010		<b>STA 2507H</b>	Store the carry (Accumulator content) in 2507H
2013		<b>HLT</b>	Stop the program

# CALL Instruction

## Unconditional CALL

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
CALL addr (label) $[SP]-1 \leftarrow [PCH]$ , $[[SP]-2 \leftarrow [PCL]$ , $[SP] \leftarrow [SP]-2$ , $[PC] \leftarrow \text{addr}(\text{label})$	Unconditional CALL: Call the subroutine identified by the address	18	None	Immediate /register	5

## Conditional CALL

Instruction Set	Explanation	States	Machine Cycles
CALL addr (label) $[SP]-1 \leftarrow [PCH]$ , $[[SP]-2 \leftarrow [PCL]$ , $[PC] \leftarrow \text{addr}(\text{label})$ , $[SP] \leftarrow [SP]-2$	Unconditional CALL: Call the subroutine identified by the address if the specified condition is fulfilled	18, if true and 9, if not true	5, if true and 2, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
CC addr(label)	Call subroutine if carry status CS=1	CS =1	9/18	None	Immediate /register	2/5
CNC addr (label)	Call subroutine if carry status CS=0	CS =0	9/18	None	Immediate /register	2/5
CZ addr (label)	Call Subroutine if the result is zero	Zero status Z=1	9/18	None	Immediate /register	2/5
CNZ addr (label)	Call Subroutine if the result is not zero	Zero status Z=0	9/18	None	Immediate /register	2/5
CP addr (label)	Call Subroutine if the result is plus	Sign status S=0	9/18	None	Immediate /register	2/5
CM addr (label)	Call Subroutine if the result is minus	Sign status S=1	9/18	None	Immediate /register	2/5
CPE addr(label)	Call subroutine if even parity	Parity Status P=1	9/18	None	Immediate /register	2/5
CPO addr(label)	Call subroutine if odd parity	Parity Status P= 0	9/18	None	Immediate /register	2/5

# RETURN Instruction

## Unconditional Return

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
RET $[PCL] \leftarrow [[SP]]$ , $[PCH] \leftarrow [[SP] + 1]$ , $[SP] \leftarrow [SP] + 2$	Unconditional RET: Return from subroutine	10	None	Indirect	3

## Conditional Return

Instruction Set	Explanation	States	Machine Cycles
RET $[PCL] \leftarrow [[SP]]$ , $[PCH] \leftarrow [[SP] + 1]$ , $[SP] \leftarrow [SP] + 2$	Conditional RET: Return from subroutine	12, if true and 6, if not true	3, if true and 1, if not true

Instruction Set	Explanation	Status	States	Flags	Addressing	Machine Cycles
RC	Return from subroutine if carry status is zero.	CS = 1	6/12	None	Register indirect	1/3
RNC	Return from subroutine if carry status is not zero.	CS = 0	6/12	None	Register indirect	1/3
RZ	Return from subroutine if result is zero.	Zero status Z=1	6/12	None	Register indirect	1/3
RNZ	Return from subroutine if result is not zero.	Zero status Z= 0	6/12	None	Register indirect	1/3
RP	Return from subroutine if result is not plus.	Sign Status S= 0	6/12	None	Register indirect	1/3
RM	Return from subroutine if result is not minus.	Sign Status S= 0	6/12	None	Register indirect	1/3
RPE	Return from subroutine if even parity.	Parity Status P= 1	6/12	None	Register indirect	1/3
RPO	Return from subroutine if odd parity.	Parity Status P= 1	6/12	None	Register indirect	1/3

# RESTART Instruction

## Restart

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
RST [[SP]-1] ← [PCH], [[SP]-2] ← [PCL], [SP] ← [SP] - 2, [PC] ← 8 times n	Restart is a one word CALL instruction.	12	None	Register Indirect	3

The restart instructions and locations are as follows:

Instruction	Opcode	Restart Locations
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

## PCHL

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
PCHL [PC] ← [H-L], [PCH] ← [H], [PCL] ← [L]	Jump address specified by H-L pair	6	None	Register	1

# Stack, I/O and Machine Control Group

- This group contains the instructions for input/output ports, stack and machine control.

Instruction Set	Explanation	States	Flags	Addressing	Machine Cycles
IN port - address $[A] \leftarrow [\text{Port}]$	Input to accumulator from I/O port	10	None	Direct	3
OUT port- address $[\text{Port}] \leftarrow [A]$	Output from accumulator to I/O port	10	None	Direct	3
PUSH rp $[[SP] - 1] \leftarrow [rh],$ $[[SP] - 2] \leftarrow [rh],$ $[SP] \leftarrow [SP] - 2$	Push the content of register pair to stack	12	None	Register(source)/register Indirect(destination)	3



PUSH PSW [SP]-1] ← [A], [[SP] -2] ← PSW, [SP] ← [SP] - 2	Push processor word	12	None	Register(source)/register Indirect(destination)	3
POP rp [rl] ← [ [ SP ] ], [rh] ← [[SP]+1], [SP] ← [SP] + 2	Pop the content of register pair, which was saved, from the stack	10	None	Register(source)/register Indirect(destination)	3
HLT	Halt	5	None		1
XTHL [L] ↔ [[SP]], [H] ↔ [[SP] + 1]	Exchange top stack with H-L	16	None	Register indirect	5
SPHL [H-L] → [SP]	Moves the contents of H-L pair to stack pointer	6	None	Register	1
EI	Enable Interrupts	4	None		1
SIM	Set Interrupts Masks	4	None		1

SIM	Set Interrupts Masks	4	None		1
RIM	Read Interrupts Masks	4	None		1
NOP	No Operation	4	None		1

# PROGRAM 7

**Ques: WAP to find larger of two numbers and store it in 2503H**

**Assume the first no. 98H is placed in the memory location 2501H**

**And the second no. 87H is placed in the memory location 2502H**

Address	Label	Mnemonics & Operands	Comments
2000H		LXI H 2501	Address of 1 <sup>st</sup> no in HL pair
2003H		MOV A,M	1 <sup>st</sup> no in accumulator
2004H		INX H	Address of 2nd no in HL pair
2005H		CMP M	Compare 2 <sup>nd</sup> no with 1 <sup>st</sup> no (Is A > M)
2006H		JNC AHEAD	Yes, large no is in accumulator, goto ahead
2009H		MOV A,M	No, get second no in accumulator
200AH	AHEAD:	STA 2503H	Store large no in 2503H
200DH		HLT	stop

# PROGRAM 8

Ques: WAP to find the largest number in a data array. Assume the numbers are 98H, 75H, 88H, 99H stored in the memory locations 2501H to 2504H and store the result in 2505H.

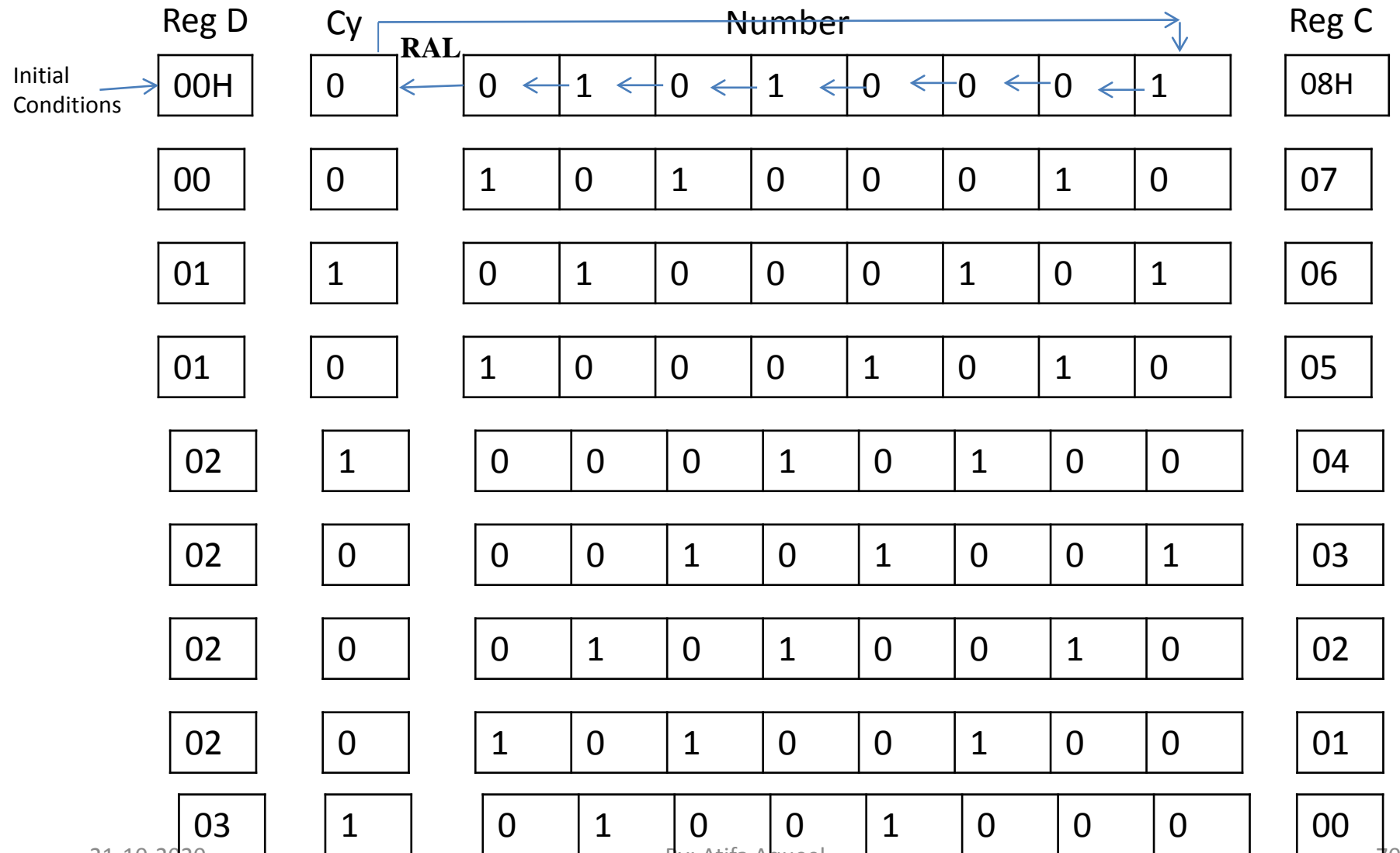
Address	Label	Mnemonics & Operands	Comments
2000		MVI C,04H	Take count in Reg C
2002		MVI A,00H	Get 00H in accumulator
2004		LXI H 2501H	Address of 1 <sup>st</sup> no in HL pair (M=98H)
2007	LOOP:	CMP M	Compare the no present in memory with the no present in accumulator. Is A>M ?
2008		JNC AHEAD	Yes, large no is in accumulator, goto ahead
200B		MOV A,M	No, get second no in accumulator
200C	AHEAD:	INX H	Get address of 2nd no in HL pair
200D		DCR C	Decrement Count (C=C - 1)
200E		JNZ LOOP	If C $\neq$ 0, Jump to LOOP
2011		STA 2505H	If C = 0, Store the result in 2505
2014		HLT	Stop

# PROGRAM (H.W)

Ques: WAP to find the smallest number in a data array. Assume the numbers are 98H, 75H, 88H, 99H stored in the memory locations 2501H to 2504H and store the result in 2505H.

# PROGRAM 9

**Q: WAP to count the number of one's present in a byte.**



# Solution

Address	Label	Mnemonics & Operands	Comments
2000H		MVI D,00H	Initialize reg D for storing the result (D=00H)
2002H		MVI C, 08H	Take count in Reg C (C=08H)
2004H		LDA 2501H	Load the 8 bit number in accumulator from [2501H]
2007H	LOOP:	RAL	Rotate the content of accumulator left with carry
2008H		JNC AHEAD	Check carry flag. If CY=0, goto AHEAD
200BH		INR D	If CY=1, Increment register D by 1
200CH	AHEAD :	DCR C	Decrement the count (Register C) by 1
200DH		JNZ LOOP	If C $\neq$ 0, Jump to LOOP
2010H		MOV A,D	If C = 0, Move the result (content of Reg D) to Accumulator
2011H		STA 2502H	Store the result in 2502H
2014H		HLT	Stop the program

# PROGRAM (H.W)

- Ques : WAP to find the number of zeroes present in a byte stored in memory location 2505H and store the result in memory location 2506H.



# PROGRAM 10

Q: WAP to perform decimal addition of two 8-bit no. present in memory location 2501H and 2502H and store the result in 2503H and 2504H.

Address	Label	Mnemonics & Op	Comments
2000H		XRA A	Clear Accumulator
2001H		MOV C,A	Move Accumulator content (i.e. 00H) to reg C
2002H		LDA 2501H	Load the content of 2501H (i.e. one operand) to Accumulator
2005H		MOV B,A	Move Accumulator content (i.e. one operand) to reg B
2006H		LDA 2502H	Load the content of 2502H (i.e. other operand) to Accumulator
2009H		ADD B	Add the content of A and B and store the result in A
200AH		DAA	Convert the result in decimal form
200BH		JNC AHEAD	Jump to AHEAD if CY = 0
200EH		INR C	If CY=1, increment Reg C by 1, C=1
200FH	AHEAD :	STA 2503H	Store the result (Accum) in 2503H
2012H		MOV A,C	Move content of C (i.e. Carry) in Accum.
2013H		STA 2504H	Store the Carry (Accum) in 2504H
2016H		HLT	Stop the program

# PROGRAM 11

**Ques: WAP to multiply two 8-bit number by successive addition method.**

**Assume :**

**2501H : 1<sup>st</sup> number (03H)**

**2502H: 2<sup>nd</sup> number (F2H)**

**2503H: LSB of result**

**2504H: MSB of result**

# SOLUTION

Address	Label	Mnemonics & Operands	Comments
		MVI C,00H	Initialize Register C for storing MSB of result
		LDA 2501	Load the content of 2501H (i.e. one operand) to Accumulator
		MOV B,A	Move Accumulator content (i.e first operand) to reg B
		LDA 2502H	Load the content of 2501H (i.e. other operand) to Accumulator
		MOV D,A	Move Accumulator content (i.e second operand) to reg D
		MVI A,00H	Initialize Accumulaor (A=00H)
	LOOP:	ADD D	Add the content of A and D and store the result in A
		JNC AHEAD	Jump to AHEAD if CY = 0
		INR C	If CY=1, increment Reg C by 1
	AHEAD:	DCR B	Decrement the count (Register B or First no.) by 1
		JNZ LOOP	If C $\neq$ 0, Jump to LOOP
		STA 2503H	If C = 0, Store the LSB of result (Acc. content) in 2503H
		MOV A, C	Move the MSB of result (content of Reg C) to Accumulator
		STA 2504H	Store the MSB of result (Acc. content) in 2504H
21-10-2020		HLT	Stop the Program

# PROGRAM 12

- **Ques: WAP to perform division of two 8-bit number by successive subtraction method.**
- 2501H– Divisor (03H)
- 2502H– Dividend (08H)
- 2503H– Quotient
- 2504H– Remainder

Initialize Reg C (C=00H)

1. CMP B : A>B, Cy flag=0  
A – B = 08 – 03 = 05H, C = 00 + 01 = 01H and A = 05H
2. CMP B : A>B, Cy flag=0  
A – B = 05 – 03 = 02H, C = 01 + 02 = 02H and A = 02H
3. CMP B : A<B, Cy flag=1  
store the result  
Quotient = Reg C and Remainder = Reg A

Address	Label	Mnemonics & Operands	Comments
2000		MVI C,00H	Initialize Register C for storing quotient
2002		LXI H 2501H	Load the value 2501H in HL pair (HL =2501H)
2005		MOV B,M	Move the content of memory location whose address is in HL pair to register <b>B</b> . (B=03H: Divisor)
2006		INX H	Increment HL pair by 1 (i.e, HL=2502H)
2007		MOV A,M	The value of 2501H will be moved to Accumulator (A=08H: Dividend)
2008	BACK:	CMP B	Compare the content of Reg A and Reg B (Is A> B?)
2009		JC AHEAD	If No (A<B) then Cy flag is 1 , goto AHEAD
200C		SUB B	If No (A>B) then Cy flag is 0, Subtract A and B
200D		INR C	Increment the value of Reg C y 1 (C = C+1)
200E		JMP BACK	Jump to the label BACK
2011	AHEAD :	INX H	Increment HL pair by 1 (i.e, HL=2503H)
2012		MOV M,C	Store the value of C (Quotient) in 2503H
2013		INX H	Increment HL pair by 1 (i.e, HL=2504H)
2014		MOV M,A	Store the value of A (Remainder) in 2504H
2015	21-10-2020	HLT	Stop the program

# HOME ASSIGNMENT (Programs)

1. WAP to find one's complement of a 16 bit number stored at memory locations 2501H and 2502H.
2. WAP to find two's complement of an 8 bit number stored at memory location 2501H
3. WAP to shift an 8 bit no. left by one bit.
4. WAP to shift a 16 bit no. left by one bit.
5. Write an assembly language program to subtract two 8- bit hexadecimal number.
6. Write an assembly language to replace all bytes from memory location 2051H to 205AH by 00H.
7. Write an assembly language to perform addition of a series of 8-bit numbers stored in the memory location 2051H to 2055H and store the results in memory location 2056H and 2057H.

THANK YOU