Practical – 5

Aim: Write an Assembly Language Program to perform the addition of two 8-bit numbers.

ADD
The content of operand are added to the content of the accumulator and the result is stored in Accumulator.

→ **ADD R** is a mnemonic that stands for "Add contents of R to Accumulator".

→ As addition is a binary operation, so it requires two operands to be operated on.

→ So input operands will reside on Accumulator and R registers and after addition the result will be stored back on to Accumulator.

→ In this case, "R" stands for any of the following registers or memory location M pointed by HL pair.

R = A, B, C, D, E, H, L, or M

→ It is 1-Byte instruction so occupies only 1-Byte in memory. As R can have any of the eight values, there are eight opcodes for this type of instruction.

Mnemonics, Operand	Bytes
ADD A	1
ADD B	1
ADD C	1
ADD D	1
ADD E	1
ADD H	1
ADD L	1
ADD M	1

ADD	R	A = A + R	ADD B
ADD	M	A = A + Mc	ADD 2050

MOV: Copy the data from one place to another.

MOV Rd, Rs

→ Copies the content of Rs to Rd (MOV 1-byte instruction)

MOV M, Rs

→ **MOV M, r** will copy 8-bit value from the register r to the memory location as pointed by HL register pair. This instruction uses register addressing for specifying the data.

→ As "r" can have any one of the seven values −

```
r = A, B, C, D, E, H, or L
```

→ Copies the content of register Rs to memory location pointed by HL Register

→ 2-byte instruction

→ Copy the data byte from the register in to memory specified by the address in HL register.

→ Thus, there are seven opcodes for this type of instruction. It occupies only 1-Byte in memory.

Mnemonics, Operand	Bytes
MOV M, A	1
MOV M, B	1
MOV M, C	1
MOV M, D	1
MOV M, E	1
MOV M, H	1
MOV M, L	1

MOV Rd, M

→ Copies the content of memory location pointed by the HL register to the register Rd.

→ 2 byte instruction

→ Copy the data byte in to the register from the memory specified by the address in HL register.

→ **MOV r, M** is an instruction where the 8-bit data content of the memory location as pointed by HL register pair will be moved to the register r. Thus this is an instruction to load register r with the 8-bit value from a specified memory location whose 16-bit address is in HL register pair.

→ As r can have any of the seven values, there are seven opcodes for this type of instruction.

```
r = A, B, C, D, E, H, or L
```

Mnemonics, Operand	Bytes
MOV A, M	1
MOV B, M	1
MOV C, M	1
MOV D, M	1
MOV E, M	1
MOV H, M	1
MOV L, M	1

Examples:
MVI B, 10h
MOV A, B
MOV M, B
MOV C, M

MVI: Move immediate data to a register or memory location.

→ **MVI** is a mnemonic, which actually means "Move Immediate". With this instruction,we can load a register with an 8-bitsor 1-Bytevalue.
→ This instruction supports immediate addressing mode for specifying the data in the instruction.
→ In the instruction "d8" stands for any 8-bit data, and 'r' stands for any one of the registers e.g. A, B, C, D, E, H or L. So this r can replace any one of the seven registers.
→ As 'r' can have any of the seven register names, so there are seven opcodes for this type of instruction. It occupies 2-Bytes in the memory.

Mnemonics, Operand	Bytes
MVI A, Data	2
MVI B, Data	2
MVI C, Data	2
MVI D, Data	2
MVI E, Data	2
MVI H, Data	2
MVI L, Data	2

MVI Rd, #30H
→ 30h is stored in register Rd
MVI M, #30H
→ 30h is stored in memory location pointed by HL Reg
→ 2 byte instruction
Examples:
MVI B, 10H [Loads the 8 bits of the 2nd byte in to the register specified]
MVI M, 30H

LDA: Load Accumulator
(This instruction copies the data from a given 16 bit address to the Accumulator)

→ **LDA** is a mnemonic that stands for Load Accumulator with the contents from memory.
→ In this instruction Accumulator will get initialized with 8-bit content from the 16-bit memory address as indicated in the instruction as a16.
→ This instruction uses absolute addressing for specifying the data.
→ It occupies 3-Bytes in the memory.
→ First Byte specifies the opcode, and the successive 2-Bytes provide the 16-bit address, i.e. 1-Byte each for each memory location.
→ 3 byte instruction

LDA 3000H
→ Load the data byte from Memory into Accumulator specified by 16 bit address
→ Content of memory location 3000h is copied in accumulator

Example:1
start: nop
LDA var1
hlt
var1: db 04h

Example:2 (Store 45H data to Specific Memory Location 000BH)
start: nop
lxi h, 000Bh
MVI M, 45H
LDA 000Bh
Hlt

LXI: (Load register pair immediate)
The instruction loads 16-bit data in the register pair designated in the operand.

LXI Rp, 16 bit
→ 3 byte instruction
→ Load intermediate immediate 16 bit no. in a register pair
→ These instructions are used to load the **16-bit** address into the register pair.
→ We can use this instruction to load data from memory location using the memory address, which is stored in the register pair **rp**.
→ The rp can be BC, DE, HL or SP.

→ The LXI instructions and their Hex-codes are as follows.

Mnemonics, Operand	Bytes
LXI B	3
LXI D	3
LXI H	3
LXI SP	3

Example:
LXI B,2050H

STA
The content of accumulator is copied into the memory location.

STA 16 BIT

→ **STA** is a mnemonic that stands for STore Accumulator contents in memory.
→ In this instruction, Accumulator 8-bit content will be stored to a memory location whose 16-bit address is indicated in the instruction as a16.
→ This instruction uses absolute addressing for specifying the destination.
→ This instruction occupies 3-Bytes of memory.
→ First Byte is required for the opcode, and next successive 2-Bytes provide the 16-bit address divided into 8-bits each consecutively.
→ 3 byte instruction
→ Load the data byte from A into the memory specified by 16 bit address

STA 2060H

Example:
start: nop
MVI A, 45h
STA 000Bh
Hlt

INX Rp

INX is a mnemonic that stands for "Increment extended register" and **rp** stands for register pair. And it can be any one of the following register pairs.

```
rp = BC, DE, or HL
```

→ 2 byte instruction
→ This instruction will be used to add 1 to the present content of the rp. And thus the result of the incremented content will remain stored in rp itself.
→ Though it is an arithmetic instruction, note that, flag bits are not at all affected by the execution of this instruction.

→ A register pair is generally used to store 16-bit memory address.
→ If flag bits got affected during increment of a memory address, then it may cause problems in many cases.
→ So as per design of 8085, flag bits are not getting affected by the execution of this instruction **INXrp**.
→ As rp can have any one of the three values, there are three opcodes for this type of instruction. It occupies only 1-Byte in memory.

Mnemonics, Operand	Bytes
INX B	1
INX D	1
INX H	1

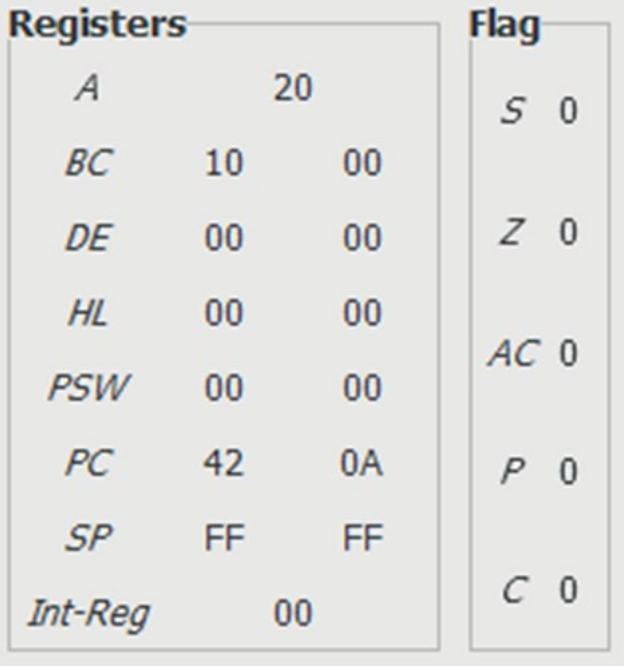
→ Example:
start: nop
LXI B, 4523h
INX B
Hlt

Opcode	Operand	Meaning	Explanation
ADD	R M	Add register or memory, to the accumulator	The contents of the register or memory are added to the contents of the accumulator and the result is stored in the accumulator. **Example** − ADD K.
INX	R	Increment register pair by 1	The contents of the designated register pair are incremented by 1 and their result is stored at the same place. **Example** − INX K

For addition of two 8-bit numbers :

```
;<Program title>

jmp start

;data

;code

start: nop

MVI  A,10H

MVI  B,10H

ADD  A

Hlt
```

Output:



For addition of two 8-bit numbers by storing address:

;<Program title>

jmp start

;data

;code

start: nop

LDA 0001H

MOV B,A

LDA 0002H

ADD A

STA 0003H

Hlt

Output:

Registers

A	14	
BC	0A	00
DE	00	00
HL	00	00
PSW	00	00
PC	42	10
SP	FF	FF
Int-Reg	00	

Flag

S	0
Z	0
AC	1
P	1
C	0

Address (Hex)	Address	Data
0000	0	0
0001	1	10
0002	2	10
0003	3	20
0004	4	0
0005	5	0
0006	6	0
0007	7	0
0008	8	0
0009	9	0
000A	10	0
000B	11	0
000C	12	0

Add two 8-bit numbers

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

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

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