MP Sessional
Lab - 01

1. Install Assembly Language Compiler (emu8086) and get familiar with it.
2. Basic Structure –

```
.MODEL SMALL      //for code size
.STACK 100H       //memory
.DATA
.CODE

MAIN PROC

        MOV AH,4CH      }     PROGRAM HAS RETURNED CONTROL
        INT 21H                TO THE OPERATING SYSTEM


        MAIN ENDP
END MAIN
```

3. Input/Output –

```
.MODEL SMALL      //for code size
.STACK 100H       //memory
.DATA
.CODE

MAIN PROC

          MOV AH,1       ;input
          INT 21H

          MOV AH,2       ;output
          MOV DL,AL       ;keyboard input automatically stored in AL register
          ;MOV DL,'M'     ;DL register used for output
          INT 21H

          MOV AH,4CH
          INT 21H


        MAIN ENDP
END MAIN
```

MP Sessional
Lab - 01

4. Variables –

💡 **Syntax:**

```asm
var_name   DB   value
```

Where:

- `var_name` = name of the variable
- `DB` = Define Byte (1 byte)
- `value` = initial value (like `'A'`, `10`, `'X'`)

Directive	Meaning
`DB`	Define Byte (1 byte)
`DW`	Define Word (2 bytes)
`DD`	Define Double Word (4 bytes)
`DS`	Data Segment Register
`MOV`	Move data into a register or memory

Code:

```
.MODEL SMALL
.STACK 100H

.DATA
char1 DB 'X'                ; variable to hold a character

.CODE
MAIN PROC
   MOV AX, @DATA
   MOV DS, AX        ; initialize data segment

   MOV DL, char1     ; load char1 into DL
   MOV AH, 02H
   INT 21H           ; print the character

   MOV AH, 4CH
   INT 21H        ; exit
MAIN ENDP
END MAIN
```

MP Sessional
Lab - 01

5. String Output -

```
.MODEL SMALL      //for code size
.STACK 100H      //memory
.DATA
msg DB 'Hello, world!$'   ;String must end with $ sign

.CODE
MAIN PROC
    MOV AH, 09H       ;09 for string output
    LEA DX, msg       ; Load address of string
    INT 21H           ; Display string

    MOV AH,4CH
    INT 21H
    MAIN ENDP
END MAIN
```

6. BASIC Instructions

1. Data Transfer Instructions

Used to **move data** between registers, memory, and immediate values.

Instruction	Description	Example
`MOV`	Copy data from one place to another	`MOV AX, 5`
`XCHG`	Exchange values between two operands	`XCHG AX, BX`
`LEA`	Load the address of a variable	`LEA DX, msg`

MP Sessional
Lab - 01

➕ 2. Arithmetic Instructions

Used to perform calculations.

Instruction	Description	Example
`ADD`	Add two values	`ADD AX, BX`
`SUB`	Subtract	`SUB AX, 10`
`INC`	Increment (add 1)	`INC CX`
`DEC`	Decrement (subtract 1)	`DEC CX`
`MUL`	Unsigned multiply	`MUL BL` (AX = AL × BL)
`DIV`	Unsigned divide	`DIV CL`

🔁 3. Logical Instructions

Used for bit-level operations.

Instruction	Description	Example
`AND`	Bitwise AND	`AND AL, 0Fh`
`OR`	Bitwise OR	`OR AL, 0Fh`
`XOR`	Bitwise XOR	`XOR AL, AL` (clears AL)
`NOT`	Bitwise NOT (inverts bits)	`NOT AL`

MP Sessional
Lab - 01

🔍 4. Comparison and Jump Instructions

Used for **decision making**.

Instruction	Description	Example
`CMP`	Compare two values	`CMP AX, BX`
`JMP`	Unconditional jump	`JMP start`
`JE / JZ`	Jump if equal / zero	`JE done`
`JNE / JNZ`	Jump if not equal / not zero	`JNE loop`
`JL / JG`	Jump if less / greater	`JL lessLabel`

🔚 7. Program Control Instructions

Instruction	Description
`INT 21H`	Call DOS service
`RET`	Return from procedure
`CALL`	Call a procedure
`HLT`	Stop the processor
`NOP`	Do nothing (No Operation)

MP Sessional
Lab - 01

7. STACK segment –

📚 What is the Stack Segment?

The **stack segment** is a special area of memory used to **store data temporarily**, following the **LIFO** principle:

> **LIFO = Last In, First Out**

So, the last item pushed into the stack will be the first one popped out.

📦 Why Use the Stack?

- Store **return addresses** for function calls
- Store **register values temporarily**
- Store **parameters and local variables**
- Allow for **nested function calls**

🧠 Important Registers:

Register	Purpose
`SS`	Stack Segment register
`SP`	Stack Pointer (points to top of stack)
`BP`	Base Pointer (often used to access stack variables)

MP Sessional
Lab - 01