LAMISA TASNIM
ID: 181147725

Homework - 3

Lecture - 1

Introduction to Assembly Language

Introduction:

In this session, you will be introduced to assembly language programming & to the emu8086 emulator software. emu8086 will be used as both an editor & as an assembler for all you assembly language programming.

Steps required to run an assembly program:

1. Write the necessary assembly source code
2. Save the assembly source code
3. Compile/assemble source code to create machine code
4. Emulate/run the machine code.

First, familiarize yourself with the software before you begin to write any code. Follow the in-class instructions regarding the layout of emu8086.

LAMISA TASNIM
ID: 1811472642

MICROCONTROLLERS vs. MICROPROCESSORS:

A microprocessor is a CPU on a single chip.

If a microprocessor, its associated support circuitry, memory & peripheral I/O components are implemented on a single chip, it is a microcontroller.

Features of 8086:

8086 is a 16-bit processor. Its ALU, internal registers work with 16 bit binary word.

8086 has a 16 bit data bus. It can read or write data to a memory/port either 16 bits or 8 bits at a time.

8086 has a 20 bit address bus which means, it can address upto 2^{20} = 1 MB memory location.

Registrar — Register — Resistor

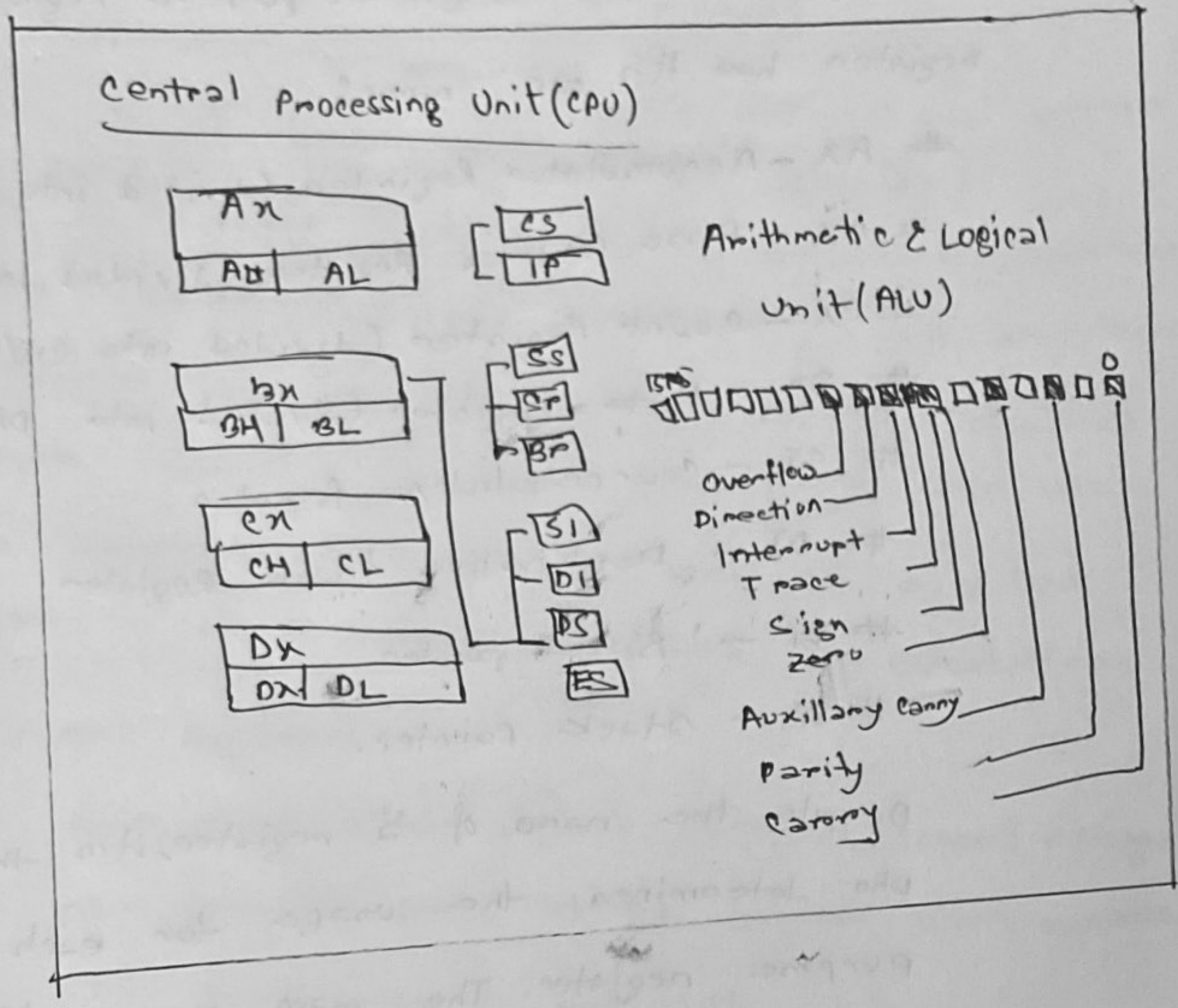
Both ALU & FPU have very small amount of super-fast private memory placed right next to them for their exclusive use. These are called registers.

The ALU & FPU store intermediate & final results from their calculations in these registers.

Processed data goes back to the data cache & then to the main memory from these registers.

LAMISA TASNIM
ID: 1811472642

Inside the CPU: Get to know the various registers



Registers are basically the CPU's own internal memory. They are used, among other purposes, to store temporary data while performing calculations. Let's look at each one in detail.

LAMISA TASNIM
ID: 1811472642

General purpose Register (GPR)

The 8086 CPU has 8 general purpose registers; each register has it's own name:

AX - Accumulator Register (divided into AH/AL)
BX - Base Address Register (divided into BH/BL)
CX - Count Register (divided into CH/CL)
DX - Data Register (divided into DH/DL)
SI - Source Index Register
DI - Destination Index Register
BP - Base pointer
SP - Stack pointer.

Despite the name of a register, it's the programmer who determines the usage for each general-purpose register. The main purpose of a register is to keep a number (variable). The size of the above registers is 16 bit.

4 general-purpose registers (AX, BX, CX, DX) are made of two separates 8-bit registers, for example if AX = 0011000000111001 b, then AH = 00110000b & AL = 00111001b. . . when you modify any of the 8-bit registers 16-bit registers are also updated & vice-versa. The name is for other 3 registers, "H" is for high & "L" is for low part.

LAMISA TASNIM
ID: 1811472642

Since registers are located inside the CPU, they are much faster than a memory. Accessing a memory location requires the use of a system bus, so it takes much longer. Accessing data in a register usually takes no time. Therefore, you should try to keep variables in the registers. Register sets are very small & most registers have special purposes which limit their use as variables, but they are still an excellent place to store temporary data of calculations.

Segment Registers

CS - points at the segment containing current program

DS - generally points at the segment where variable

ES - Extra segment register

SS - points at the segment containing the stack

Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have a special purpose - pointing at accessible blocks of memory. This will be discussed further in upcoming classes.

LAMISA TASNIM
ID: 1811472642

Special Purpose Registers

#IP - The instruction pointer. Points to next location of instruction in the memory.

Flag register - determines the current state of the microprocessor. Modified automatically by the CPU after some mathematical operations, determines certain types of results & determines how to transfer control of a program.

Writing Your First Assembly Code:

In order to write programs in assembly language, you will need to familiarize yourself with most, if not all, of the instructions in the 8086 - instruction set. This class will introduce two instructions & will serve as the basis for your first assembly program.

The following table shows the instruction name, the syntax of its use, & its description. The operands heading refers to the type of operands that can be used with the instruction along with their proper order.

LAMISA TASNIM
ID: 1811472642

REG: Any valid register

MEMORY: Referring to a memory location in RAM

Immediate: Using direct values.

Instruction	Operands	Description
MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Copy Operand2 to Operand1. # The MOV instruction cannot: set the value of the CS & IP registers. # Copy value of one segment register to another segment register. # copy an 'immediate' value to segment register (should copy to general register first) Algorithm operand1 = operand2
ADD	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Adds two numbers. Algorithm: operand1 = operand1 + operand2

LAMISA TASNIM
ID: 1811472642

Homework - 3

Lecture - 2

Variables, I/O, Array

Topics to be covered in this class:

Creating variables
Creating Arrays
Create constants
Introduction to INC, DEC, LEA instruction
learn how to access memory

Creating variable:

Syntax for a variable declaration:

name DB value

name DW value

DB - stands for define Byte

DW - stands for Define Word

#name - can be any letter ~~of~~ digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no ~~no~~ name).

LAMISA TASNIM
ID: 1811472642

#value - can be any numerical value in any supported numbering system (hexadecimal, binary or decimal), or "?" symbol for variables that are not initialized.

Creating Constants:

Constants are just like variables, but they exist only until your program is compiled (assembled). After definition of a constant its value cannot be changed. To define constants EQU directive is used.

name EQU <any expression>

For example:

```
k EQU 5
MOV AX, k
```

LAMISA TASNIM
ID° 1811472642

Creating Arrays:

Arrays can be seen as chains of variables.
A text string is an example of a byte array, each array is presented as an ASCII code value (0 - 255).

Here are some array definition examples:

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b DB 'Hello', 0

→ you can access the value of any element in array using square brackets, for example:

MOV AL, a[3]

→ you can also use any of the memory index registers BX, SI, DI, BP, for example:

MOV SI, 3
MOV AL, a[SI]

→ If you need to declare a large array you can use DUP operator.

LAMISA TASNIM
ID: 1811472

The syntax for DUP:

number DUP (value(s))

number - number of duplicates to make (any constant value)
value - expression that DUP will duplicate

for example:

c DB 5 DUP(9)

is an alternative way of declaring:

c DB 9,9,9,9,9

one more example:

d DB 5 DUP (1,2)

is an alternative way of declaring:

d DB 1,2,1,2,1,2,1,2,1,2

Memory Access

To access memory, we can use these four registers: BX, SI, DI, BP. Combining these registers inside [ ] symbols, we can get different memory locations.

LAMISA TASNIM
ID: 1811472642

[BX+SI] [BX+DI] [BP+SI] [BP+DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX+SI+d8] [BX+DI+d8] [BP+SI+d8] [BP+DI+d8]
[SI+d8] [DI+d8] [BP+d8] [BX+d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP+DI+d16]	[SI+d16] [DI+d16] [BP+d16] [BX+d16]

→ Displacement can be an immediate value or offset of a variable, or even both. If there are several values, assembler evaluates all values & calculates a single immediate value.

→ Displacement can be inside or outside of the [] symbols, assembler generates the same machine code for bothways.

→ Displacement is a signed value, so it can be both positive & negative.

LAMISA TASNIM
ID: 1811472642

Instructions

Instruction	Operands	Description
INC	REG MEM	Increment. Algorithm: operand = operand+1 Example: MOV AL, 4
DEC	REG MEM	Decrement. Algorithm: operand = operand-1 Example: MOV AL, 86 DEC AL ; AL = 85 RET

LEA	REG, MEM	Load Effective Address.
		Algorithm:
		REG = address of memory (offset)
		Example:
		MOV BX, 35H
		MOV DI, 12H
		LEA SI, [BX+DI]

Declaring Array:

Array Name db Size DUP (?)

Value initialize:

arr1 db 50 dup (5, 10, 12)

Index values:

```
mov bx, offset arr0
mov [bx], 6 ; inc bx
mov [bx+1], 10
mov [bx+9]_9
```

LAMISA TASNIM
ID: 1811472642

Offset:

"Offset" is an assembler directive in x86 assembly language. It actually means "address" & is a way of handling the overloading of the "mov" instruction. Allow me to illustrate the usage—

1. mov si, offset variable
2. mov si, variable

the first line loads SI with the address of variable. The second line loads SI with the value stored at the address of variable.

As a matter of style, when i wrote x86 assembler I would write it this way—

1. mov si, offset variable
2. mov si, [variable]

The square brackets aren't necessary but they made it much clearer while loading the contents rather than the address.

LEA is an instruction that load "offset variable" while adjusting the address between 16 & 32 bits as necessary. "LEA (16-bit), (32-bit)" loads the lower 16 bits of the address into the register & "LEA (32-bit), (16-bit)" loads the 32-bit register with the address zero extend to it.

LAMISA TASNIM
ID: 1811472642

Lecture - 3

Print & I/O

In this assembly language programming, A program is divided into 4 segments are:-

1. Data segment
2. Code segment
3. Stack segment
4. Extra segment

Print : Hello World in Assembly language

~~Data As~~

```
DATA SEGMENT
        MESSAGE DB 'HELLO WORLD !!! $'
ENDS
CODE SEGMENT
        ASSUME DS:DATA CS: CODE

START:
        MOV AX, DATA
        MOV DS, AX
        LEA DX, MESSAGE
        MOV AH, 9
        INT 21H
        MOV AH, 4CH
        INT 21H
ENDS
END START
```

LAMISA TASNIM

ID:1811472642

First line - DATA SEGMENT

DATA SEGMENT is the starting point of the DATA SEGMENT in a program & DATA is the name given to this segment & SEGMENT is the keyword for defining segments, where can declare our variables.

Next line - MESSAGE DB "HELLO WORLD!!!$"

MESSAGE is the variable name given to a Data Type (size) that is DB. DB stands for Define Byte & is of one byte (8 bits). In Assembly language programs, variables are defined by Data Size not its Type. Character need One Byte no to store Character or string we need DB only that don't mean DB can't hold number or numerical value. The string is given in double quotes. $ is used an NULL character in C programming

NEXT line - DATA ENDS

DATA ENDS is the End point of the Data Segment in a program. We can write just ENDS But to differentiate the end of which segment it is of which we have to write the name name given to the data segment.

LAMISA TASNIM
ID:181147262642

NEXT Line - CODE SEGMENT:

CODE SEGMENT is the starting point of the CODE Segment in a program & code is the name given to this segment & SEGMENT is the keyword for defining Segments, where we can write the coding of the program.

Next line - ASSUME DS: DATA CS: CODE :

In this Assembly language programming, there are Different Registers present for Different Purpose. So we have to assume DATA is the name given to Data segment Register & CODE is the name given to Code segment Register (SS, ES are used as the same way as CS, DS)

Next Line - START:

START is the label used to show the starting point of the code which is written in that Code Segment:: is used to define a label as in C programming.

LAMISA TASNIM
ID: 1811472642

Next Line - MOV AX, DATA
MOV DS, AX
After Assuming DATA & CODE segment, still it is compulsory to initialize Data segment to DS register. Mov is a keyword to move the second element into the first element. But we cannot move DATA Directly to DS due to MOV commands restriction, hence we move DATA to AX & then from AX to DS. AX is the first & most important register in the ALU unit. This part is also called INITIALIZATION OF DATA SEGMENT & it is important so that the Data elements or variables in the DATA Segment are made accessible. Other segments are not needed to be initialized, only assuming is inhale.

Next Line - LEA DS, MESSAGE
MOV AH, 9
INT 21H
The above three-line code is used to print the string inside the MESSAGE variable. LEA stands for Load Effective Address which is used to assign Address of variable to DX register. To do input & output in Assembly language we use Interrupts. Standard Input & Standard Output related Interrupts are found in INT 21H which is also called as DOS interrupt. If the value is 9 or 9H, PRINT the string whose Address is Loaded in DX.

LAMISA TASNIM
ID: 1811472642

Next line - MOV AH, 4CH

INT 21H

The above two-line code is used to exit to dos or exit to operating system. Standard Input & Standard Output related interrupts are found in INT 21H which is also called as DOS Interrupt. It works with the value of AH register, if the value is 4ch, that mean Return to Operating system or DOS which is the End of the program.

Next line - CODE ENDS

CODE ENDS is the End point of the Code Segment in a program. We can write just ENDS But to differentiate the end of which segment it is of which we have to write the same name given to the Code Segment.

Last Line - END START

END START is the end of the label used to show the ending print of the code which is written in the Code Segment.

LAMISA TASNIM
ID: 1811472642

Assembly Example 4 – Print 2 strings

```
. MODEL  SMALL
. STACK  100H
. DATA
    STRING_1  DB 'I hate CSE331 $'
    STRING_2  DB 'But I live kacchi !!! $'
. CODE
  MAIN PROC
    MOV  AX, @DATA   ; initialize DS
    MOV  DS, AX
    LEA  DS, STRING_1   ; load & display STRING_1
    MOV AH, 9
    INT  21H
    MOV AH, 2  ; carriage return
    MOV DL, ODH
    INT 21H
    MOV DL, 0AH ; Line feed
    INT 21H
    LEA  DX, STRING_2 ; load & display STRING_2
    MOV AH, 9
    INT 21H
    MOV AH, 4CH ; return control to DOS
    INT 21H
  MAIN ENDP
END MAIN
```

LAMISA TASNIM
ID: 18114772642

Assembly Example 2 - Read a String & Print it

```
.MODEL SMALL
.STACK 100H

.DATA
    MSG_1 EQU 'Enter the Character : $'
    MSG_2 EQU 0DH, 0AH, 'The given character is : $'

    PROMPT_1 DB MSG_1
    PROMPT_2 DB MSG_2

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

        LEA DX, PROMPT_1 ; load & display PROMPT_1
        MOV AH, 9
        INT 21H

        MOV AH, 1 ; read a character
        INT 21H
        MOV BL, AL ; save the given character in BL

        LEA DS, PROMPT_2 ; load & display PROMPT_2
        MOV AH, 9
        INT 21H
        MOV AH, 2 ; display the character
        MOV DL, BL
        INT 21H
        MOV AH, 4CH
        INT 21H
    MAIN ENDP
END MAIN
```

LAMISA TASNIM
ID: 1811472642

Assembly Example 3 - Read a string from user & display this string in a new line.

```
.MODEL SMALL
.STACK 100H

.CODE
    MAIN PROC
        MOV AH,1 ; read a character
        INT 21H
        MOV BL,AL ; save input character into BL
        MOV AH,2 ; carriage return
        MOV DL,0DH
        INT 21H
        MOV DL,0AH; line feed
        INT 21H
        MOV AH,2 ; display the character
                 stored in BL
        MOV DL,BL
        INT 21H
        MOV AH,4CH ; return control to DOS
        INT 21H
    MAIN ENDP
ENDMAIN
```

LAMISA TASNIM
ID: 1811472642

ASSembly Example 4- Read a string with gaps & print it

```
.MODEL SMALL
.STACK 64
.DATA
    STRING DB ?
    SYM DB '$'
    INPUT_M DB 0ah,0dh,0AH,0DH,'Enter the Input',0DH,0AH,$
    OUTPUT_M DB 0ah,0dh,0AH,0DH,'The output is',0DH,0AH,'$'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    MOV DX, OFFSET INPUT_M ;lea dx,input_m
    MOV AH,09
    INT 21H
    LEA SI, STRING
INPUT : MOV AH,01
    INT 21H
    MOV [SI],AL
    INC SI
    CMP AL,0DH
    JNZ INPUT
    ; MOV AL,SYM
    MOV [SI],'$'
```

LAMISA TASNIM

ID: 1811972642

```
OUTPUT : LEA   DX, OUTPUT - M   ; load & display PROMPT_2

         MOV   AH, 9

         INT 21H

         MOV   DL, 0AH

         MOV AH, 02H

         INT   21H

         MOV DX, OFFSET STRING

         MOV   AH, 09H

         INT   21H

         MOV   AH, 4CH

         INT 21H

MAIN ENDP

END MAIN
```

Assembly Example 5 – Printing string using MOV instruction

```
.Model Small
; .STACK

.DATA
        MSG1 DB 'Hi!!! Kemon lage :D $'

.CODE
        MOV   AX, @DATA
        MOV   DS, AX

        MOV   dx, OFFSET MSG1   ; LEA dx, msg1
        mov   ah, 09H
        int 21h

        mov   ah, 4Ch
        int 21h

END
```

ZAMISA TASNIM
ID : 1811472642

Assembly Example 6- Print Digit from 0-9

```
.MODEL SMALL

.STACK 100H

.DATA
    PROMPT DB \'The counting from 0 to 9 is : $\'

.CODE
    MAIN PROC
        MOV AX, @DATA ;initialize DS
        MOV DS, AX
        LEA DX, PROMPT ;load & print prompt
        MOV AH, 9
        INT 21H
        MOV CX, 10 ;initialize CX
        MOV AH, 2 ; set output function
        MOV DL,48 ; set DL with 0

@LOOP :                 ; loop label
        INT 21H         ; print character
        INC DL ; increment DL to next ASCII
                 character

        DEC CX ; decrement CX
        JNZ @LOOP ; jump to label @LOOP if CX is 0
        MOV AH, 4CH ; return control to DOS
        INT 21H
    MAIN ENDP
END MAIN
```

LAMIA TASNIM

ID : 1811472642

Assembly Example 2 – Sum of two integers

```
.MODEL SMALL
.STACK 100H
.DATA
    PROMPT_1 DB \'Enter the First digit : $\'
    PROMPT_2 DB \'Enter the Second digit : $\'
    PROMPT_3 DB \'Sum of first & second digit : $\'
    VALUE_1 DB ?
    VALUE_2 DB ?
.CODE
    MAIN PROC
        MOV AX, @DATA  ;initialize
        MOV DS, AX
        LEA DX, PROMPT_1 ; load & display PROMPT_1
        MOV AH, 9
        INT 21H
        MOV AH, 1  ; read a character
        INT 21H
        SUB AL, 30H  ; save First digit in VALUE_1
                     in ASCII code
        MOV VALUE_1, AL
        MOV AH, 2 ; carriage return
        MOV DL, 0DH
        INT21H
```

LAMISA TASNIM
ID: 1811472642

```
MOV   DL, 0A H   ; line feed
INT   21H

LEA     DX , PROMPT_2
MOV     AH_ 9
INT     21H
MOV     AH, 1 ; read a character
INT     21H

SUB       AL , 30H   ;save Second digit in VALUE_2
                      in ASCII code
MOV   AH , 2    ; carriage return
MOV     DL, 0PH
INT 21H

MOV     DL, 0AH  ; line feed
INT 21H

LEA     DX , PROMPT_3 ; load & display the PROMPT_3
MOV AH, 9
INT   21H

MOV   AL, VALUE_1 ; add First & Second digit
ADD   AL, VALUE_2
ADD   AL, 30H
MOV   AH, 2
MOV     DL, AL
INT 21H
  MOV   AH , 4CH
INT 21H
END MAIN      MAIN ENDP
```