CSE 331L
(Microprocessor Interfacing & Embedded System (Lab))

HOMEWORK 03

Submitted by:

NAME: HOSNE ARA
ID: 1632267642
SECTION: 07

Submitted to:

ASIF AHMED NELOY

Assignment-03

CSE331Lab

Lab-01

CSE331L - Introduction to Assembly language.

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

Microcontrollers Vs. Microprocessors:

- A microprocessor is a CPU on a single chip.
- If a microprocessor, its associated support circuitry, memory and peripheral I/O components are implemented on a single chips, it is a microcontroller.

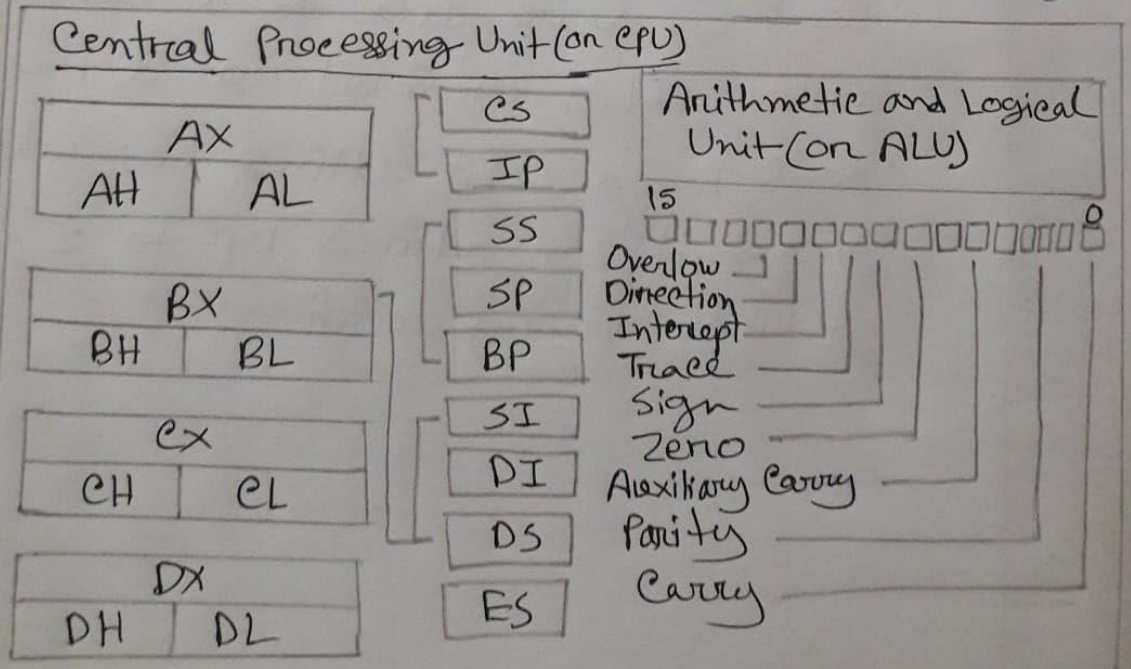
Features of 8086:

- 8086 is a 16bit processor. It's ALU, internal registers work with 16bit binary word.
- 8086 has a 16bit data bus. It can read on write data to a memory/port either 16bits on 8 bits at a time.
- 8086 has a 20bit address bus which means, it can address up to 2^20 = 1MB memory location.

Registrar - Register - Resistor:

- Both ALU and FPU have a very small amount of super-fast private memory placed right next to them for their exclusive use. These are called registers.
- The ALU and FPU store intermediate and final results from their calculations in these registers.
- Processed data goes back to the data cache and then to the main memory from these registers.

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 on in detail.

General Purpose Registers (GPR):

The 8086 CPU has 8 general-purpose registers: each register has its own name:

- AX - The Accumulator register (divided into AH/AL).
- BX - The Base Address register (divided into BH/BL).
- CX - The count register (divided into CH/CL).
- DX - The 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 bits.

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

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 and 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 the current program.
DS - generally points at the segment where variables are defined.
ES - extra segment register, it's up to a coder to define it's usage.
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 very special purpose - pointing at accessible blocks of memory. This will discussed further in upcoming classes.

Special Purpose Registers:

- IP - The Instruction Pointer. Points to the next location of instruction in the memory.
- Flags Register - Determines the current state of the microprocessor. Modified automatically by the CPU after some mathematical operations, determines certain types of results and 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 ~~int~~ instructions and will serve as the basis for your first assembly program.

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

- REGr: Any valid register.
- Memory: Referring to a memory location in RAM.
- Immediate: Using direct values.

Instruction	Operands	Description
MOV	REGr, memory memory, REGr REGr, REGr memory, immediate REGr, imediate	Copy Operand2 to Operand1. • The MOV instruction cannot: set the value of the cs and IP registers. • Copy value of one segment register to another segment register (should copy to general register first). • Copy an immediate value to segment register (should copy to general register first) Algorithm: Operand1 = Operand2
ADD	REGr, memory memory, REGr REGr, REGr memory, immediate REGr, immediate	Adds two members. Algorithm: Operand1 = Operand1 + Operand2

Lab-02

CSE331L_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 or 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 name).

• Value - can be any numeric 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 to changed. To define constants EQU directive is used:

name EQU <any expression>

For example:

```
K EQU 5
MOV AX, K
```

Creating Arrays:

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

Here are some array definition examples:

```
a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 0Dh
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.

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.

[BX + SI]	[SI]	[BX + SI + d8]
[BX + DI]	[DI]	[BX + DI + d8]
[BP + SI]	d16 (variable offset only)	[BP + SI + d8]
[BP + DI]	[BX]	[BP + DI + d8]
[SI + d8]	[BX + SI + d16]	[SI + d16]
[DI + d8]	[BX + DI + d16]	[DI + d16]
[BP + d8]	[BP + SI + d16]	[BP + d16]
[BX + d8]	[BP + DI + 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 and calculates a ~~sigle~~ single immediate value.
- Displacement can be ~~inside~~ inside or outside of the [] symbols, assembler generates the same machine code for both ways.
- Displacement is a signed value, so it can be both positive or negative.

Instructions:

Instruction	Operands	Description
INC	REG MEM	Increment. Algorithm: Operand = Operand+1 Example: MOV AL, 4

Instruction	Operands	Description
		INC AL ; AL = 5 RET
		Decrement : Algorithm: Operand = Operand-1 Example: MOV AL, 86 DEC AL, AL = 85 RET

LEA	REGr, MEM	Load Effective Address:
		Algorithm:
		REGr = 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
```

OFFSET:

"Offset" is an assembler directive in x86 assembly language. It actually means "address" and 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 make 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 and 32 bits as necessary. "LEA (16-bit register), (32-bit address)" loads the lower 16 bits of the address into the register and "LEA (32-bit register), (16-bit address)" loads the 32-bit register with the address zero extended to 32 bits.

CSE331L - Print and Lab-03 I/O

In this Assembly Language Programming. A single program is divided into four segments which are -

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

Print : Hello World in Assembly language.

```
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

Now, from these one is compulsory i.e. Code Segment if at all you don't need Variable(s) for your Program. If you need Variable(s) for your Program you will need two Segments i.e. Code Segment and Data Segment.

First Line - DATA SEGMENT

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

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

MESSAGE is the variable name given to a ~~DAT~~ Data Type (Size) that is DB. DB stands for Define Byte and is of One byte (8 bits). In Assembly language programs, Variables are defined by Data Size not its Type. Character

need One Byte so 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 as NULL character in C programming, so that compiler can understand where to STOP.

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 same name given to the Data Segment.

Next Line – CODE SEGMENT:

CODE SEGMENT is the starting point of the Code segment in a Program and CODE is the name given to this segment and SEGMENT is the Keyword for defining Segments, where we can write the Coding of the Program.

Next line - ASSUME DS: DATA CS: CODE :

In the 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 and CODE is the name given to Code Segment register (SS, ES are used in 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 the Code Segment; is used to define a label as in C programming.

Next Line - MOV AX, DATA

MOV DS, AX

After Assuming DATA and 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 and then from AX to DS. AX is the first and most important register in the ALU unit. This part is also called INITIALIZATION OF DATA SEGMENT and 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 DX, MESSAGE
MOV AH, 9
INT 21H

The above threee-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 (The same can be written like this also MOV DX, OFFSET MESSAGE both mean the same). To do input and output in Assembly Language we use Interrupts. Standard Input and 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 9 or 9h or 09H (all means the same). That means PRINT the string whos Address is loaded in DX register.

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 and 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 means 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 point of the code which is written in the Code Segment.

Execution of program explanation – Hello World

First save the program with Hello World.asm filename. No space is allowed in the name of the Program File and extension as .asm (dot asm because it's an Assembly language Program). The written Program has to be Complied

and Run by clicking on the RUN button on the top. The Program with No Errors will only ruen and could show you the desired output. Just see the screenshots below.

Note :- In this Assembly Language Programming we have Com format and EXE format. We are Learning in EXE format only which simple then Com format to understand and Write. We can write the program in lower or upper case but I prepare Upper case. (this program is executed on emu8086 emulator software).

Now Try This —

```
DATA SEGMENT
     MESSAGE DB "HELLO WORLD $"
START:
     MOV AX, DATA
     MOV DS, AX
     LEA DX, MESSAGE
     MOV AH, 9
```

```
    INT 21H
    MOV AH,4CH
    INT 21H
END START
```

Assembly Example - 1 - Print 2 strings:

```
.MODEL SMALL
.STACK 100H
.DATA
    STRING_1 DB 'I hate CSE331$'
    STRING_2 DB 'But I love kacchi!!!$'
.CODE
    MAIN PROC
        MOV AX,@DATA          ;initialize DS
        MOV DS,AX
    LEA DX, STRING_1
        MOV AH, 9             ; load & display the
        INT 21H                 STRING_1
    MOV AH, 2
    MOV DL,0DH                ; carriage return
    INT 21H
```

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

LEA DX, STRING_2     ; load & display the
MOV AH, 9                STRING_2
INT 21H

MOV AH, 4CH          ; return control to DS
INT 21H

MAIN ENDP
END MAIN
```

Assembly Example 2 - Read a String and 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 and display
MOV AH, 9                   PROMPT_1
INT 21H
MOV AH, 1                 ; read a character
INT 21H
MOV BL, AL                ; save the given character
                            into BL
LEA DX, PROMPT_2
MOV AH, 9                 ; load and display PROMPT_2
INT 21H
MOV AH, 2
MOV DL, BL                ; display the character
INT 21H
MOV AH, 4CH
INT 21H                   ; return control to DOS
MAIN ENDP
END MAIN
```

Assembly Example 3 – Read a string from user and 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
    INT 21H            ; return control to DOS
MAIN ENDP
END MAIN
```

Assembly Example 4 - Read a string with gaps and 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],'$'
OUTPUT : LEA DX,OUTPUT_M
        MOV AH,9          ;load and display
        INT 21H             PROMPT_2
        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 'KI!!! 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
```

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 and print
    MOV AH, 9              PROMPT
    INT 21H
    MOV CX, 10           ;initialize CX
    MOV AH, 2            ; set output function
    MOV DL, 48           ; set DL with 0
  @LOOP:
      INT 21H            ; loop label
                         ; 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
      INT 21H            ; return control to DOS
  MAIN ENDP
END MAIN
```

Assembly Example 7 - 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 and second digit:
                                        $'
    VALUE_1 DB ?
    VALUE_2 DB ?
.CODE
    MAIN PROC
        MOV AX, @DATA      ;initialize DS
        MOV DS, AX
    LEA DX, PROMPT_1
    MOV AH, 9              ;load and display
    INT 21H                 the PROMPT_1
    MOV AH, 1
    INT 21H                ;read a character
```

```
SUB AL, 30H
MOV VALUE_1, AL      ; save first digit in
                       VALUE_1 in ASCII code

MOV AH, 2
MOV DL, ODH          ; carriage return
INT 21H

MOV DL, OAH
INT 21H              ; line feed

LEA DX, PROMPT_2
MOV AH, 9            ; load and display
INT 21H                the PROMPT_2

MOV AH, 1
INT 21H              ; read a character

SUB AL, 30H
MOV VALUE_2, AL      ; save second digit
                       in VALUE_2 in
                       ASCII code

MOV AH, 2
MOV DL, ODH          ; carriage return
INT 21H

MOV DL, OAH
INT 21H              ; line feed
```

```
LEA DX, PROMPT_3        ;load and display
MOV AH, 9                 the PROMPT_3
INT 21H
MOV AL, VALUE-1         ;add First and Second
ADD AL, VALUE_2           digit
ADD AL, 30H             ; convert ASCII to
MOV AH, 2                 DECIMAL code
MOV DL, AL              ; display the character
INT 21H
MOV AH, 4CH             ; return control to DOS
INT 21H
MAIN ENDP
END MAIN
```