Lab 01: Introduction to Assembly Language & Basic IO Operations

Objectives:

The objectives of this lab session are to introduce the students with 80x86 based microprocessors architecture, few assembly language instructions and some IO operations using INT 21H.

After this lab you will be able to:

1. Write simple assembly language programs.
2. Assemble, Link and Run assembly source code using Emu8086
3. Take input from key board using functions call 01 of INT 21H
4. Show output on the screen using function calls 02 and 09 of INT 21H

Theory:

1.1 Overview:

Assembly language represents each of the many operations that the computer can do with a mnemonic, a short, easy to remember series of letters. For example, in Boolean algebra, the logical operation which inverts the state of a bit is called **"not"**, and hence the assembly language equivalent of the preceding machine language pattern is: **NOT AX**, where AX is the register whose contents are to be inverted. The "B" at the end of "NOTB" indicates that we want to operate on a byte of memory, not a word. Unfortunately, the CPU cannot understand the string of characters "NOTB". What is needed is a special program which converts the string "NOTB" into the machine language "1111 0110 0001 0110" represented by "F6 16" in hexadecimal form. This program is called an "ASSEMBLER". An assembler program is analogous to a machine which takes in, assembly language and gives out, machine language.

The translation from assembly language to machine code is done in following steps.

	Action	**Result**
1	Editing	Source Files (*.asm)
2	Assembling	Object files (*.obj)
3	Linking	Executable Files(*.exe)
4	Executing	Output

1.2 Assembling the Program:

The assembler is used to convert the assembly language instructions to machine code. It is used immediately after writing the Assembly Language program. The assembler starts by checking the syntax, or validity of the structure, of each instruction in the source file. If any error is found, assembler displays a report on these errors along with a brief explanation of their nature .However if the program does not contain any error, The assembler produces an object file that has the same name as the original file but with the **".obj"** extension

1.3 Linking the Program:

The linker is used to convert the object file to an executable file. The executable file is the final set of machine instructions that can directly be executed by the microprocessor. It is different than the object file in the sense that it is self contained and re-locatable. An object file may represent one segment of a long program; this segment cannot operate by itself, and must be integrated with other object files representing rest of the program, in order to produce the final self contained executable file.

In addition to the executable file, the linker can also generate a special file called the **"map"** file .This file contains information about the Start, End, and the Length of the Stack, Code and the Data segments. It also lists the Entry point of the program.

1.4 Executing the Program:

The executable file contains the machine language code. It can be loaded in the RAM and be executed by the microprocessor simply by typing, from the DOS prompt, The name of the file followed by the Enter Key(<┘). If the program produces an output on the screen, or a sequence of control signals to control a piece of hardware, the effect should be noticed almost immediately. However, if the program manipulates data in memory, nothing would seem to happen as a result of executing the program.

1.5 Debugging the Program:

The debugger can also be used to find logical errors in the program .Even if a Program does not contain syntax errors it may not produce the desired results after execution. Logical errors may be found by tracing the action of the program .Once found, the source file should be re-edited to fix the problem, then re-assembled and re-linked. A special program called the "Debugger" is designed for this purpose. The debugger allows the user to trace the action of the program, by **single stepping** through the program or executing the program up to a desired point, called **"breakpoint**". It also allows the user to inspect or change the contents of the microprocessor internal registers or the contents of any memory location.

The details of a few assembly language mnemonics that we will be using in this lab is written as under,

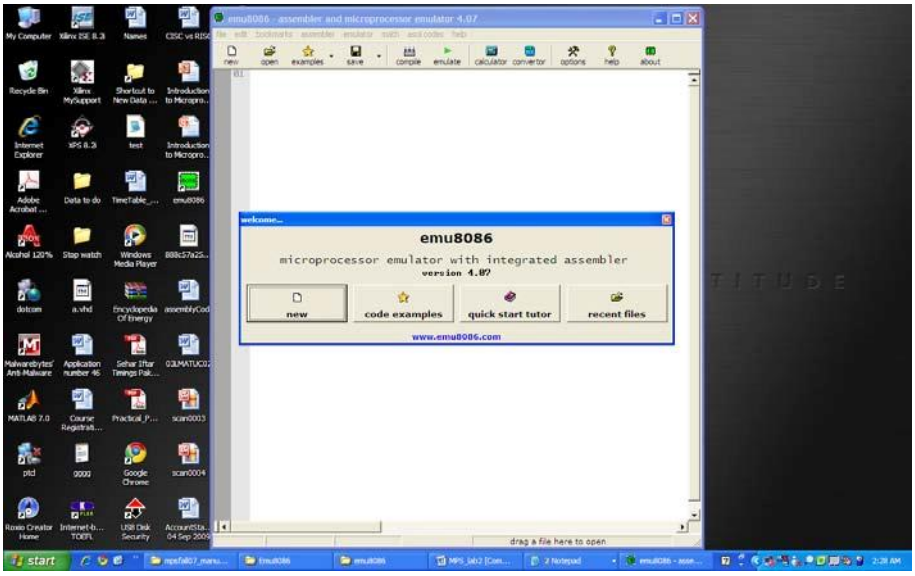
	COMMAND	FILE NAME
1	**MOV** AX,BX	Copies the contents of accumulator register BX to AX
2	**ADD** AX,BX	Adds the contents of AX and BX and saves result in AX
3	**INT** 21H	Generates interrupt No: 21 H , Which returns the control to DOS if value of register AH=4C H (more about this in future labs)

1.6 Assembling Program in Emu8086

Emu8086 is an integrated development environment; it consists of an editor, an assembler, linker and a debugger. For assembling program in EMU 8086, follow the steps given in procedure.

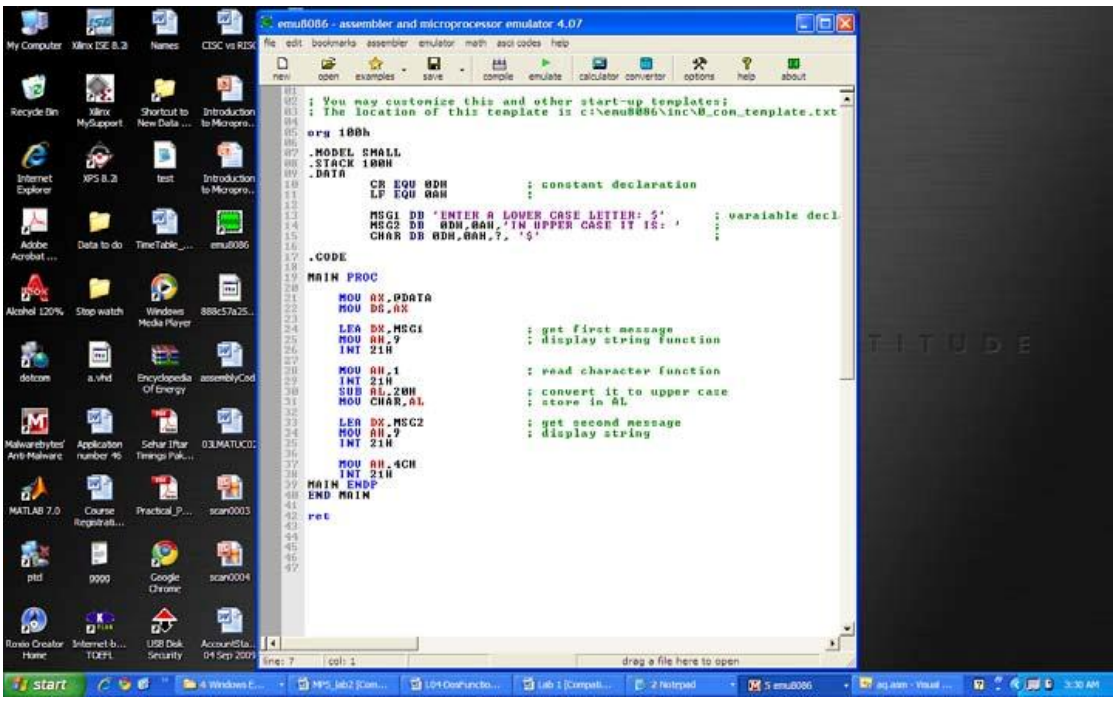
Procedure:

1. Open the emu8086 present on your desktop. Following window will appear,

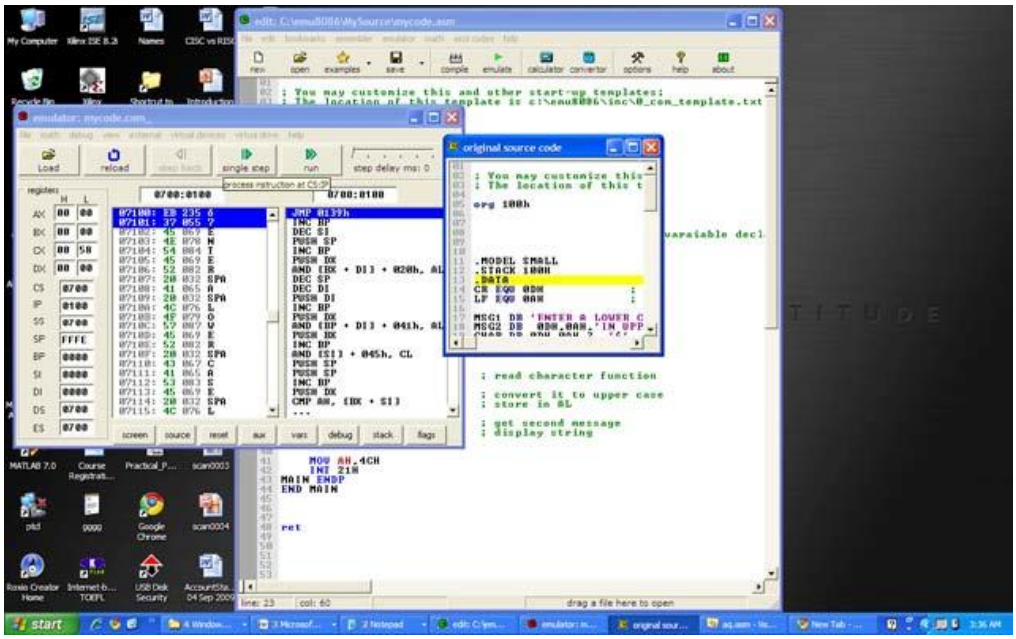


2. Select “new” from the above window. A new small window will appear select “COM Template” and press Ok. Copy the following code below the default lines.

```
.MODEL SMALL
.STACK100H
.CODE
MOV AX, 2000
MOV BX, 2000H
MOV CX, 1010001B
MOV DX, 4567
MOV BH,'A'
MOV CL,'a'
MOV AH, 4CH
INT 21H        ; INT 21h returns the control to DOS if AH=4CH
END
```



3. After writing your code, go to the file menu and save your code with an appropriate name (the .asm extension is by default, you don’t have to write it explicitly).
4. After saving your code, compile your code by pressing compile menu and view it there are any errors otherwise just press ok.
5. Now emulate your program form the menu and see the values of registers step by step



Task:

1) **Observe and write down the contents of registers AX, BX, CX and DX after the complete code is run?**

2) **Do the contents of any register change as the code is run step by step? If yes, what change is observed and in which registers?**

Input and Output operations of 8086/8088 Assembly Language

Now you will be introduced with how to perform input and output operations using DOS **INT 21H** function calls and get another step further in learning the structure of assembly language by using:

1. Variable declaration using: **DB**
2. Offset operator using **OFFSET**
3. Familiarization with friendly Emulator, **Emu8086**.

The instruction INT 21h stands for "Call Interrupt no. 21". For the time being let's take interrupts like a function or a subroutine placed in memory and being called from another program. The parameters to that function or subroutine are passed by using different CPU registers mostly AH, AL or DX. Functionality of that subroutine depends upon those parameters; Typical examples include taking input, showing output, mouse control ,VGA memory control etc. some of them are tabulated as under.

Function call No.	**Value of AH**	**Output in**	**Functionality**
01h	01h	AL	Reads a character from keyboard, stores it in AL and display it (echo it) on screen.
02h	02h	Screen	Display the content of register DL on screen in ASCII form.
09h	09h	Screen	Display the string characters addressed by DX to the screen
0Ah	0Ah	Offset in DX	Read a string of characters from keyboard.

DOS Function 02h:

To display a single ASCII character at the current cursor position, use the following sequence of instructions.

MOV AH, 02H
MOV DL, Character Code
INT 21H

The Character Code may be the ASCII code of the character taken from the ASCII table (provided along with) or the character itself written between quotes.

The following code displays number 5 using its ASCII code:

```
MOV   AH, 02H
MOV DL, 35H
INT 21H
```

Question:

What happens if we replace 35H with just 35?

Question:

What is the result of the code below?

```
MOV AH, 02H
MOV DL, '5'
INT 21H
```

Function 01H:

To read single character and have it echoed (displayed) on the screen, use the following code:

```
MOV AH, 01H
INT 21H
```

Question:

What register contains the ASCII code of the character read from the keyboard?

DOS Functions 09:

This function is used to display a string of characters ending with a '$' sign. The following code displays the string MESSAGE defined as:

```
.DATA
MESSAGE DB 'This is the Message to be displayed',

.CODE
 MOV DX, OFFSET MESSAGE
 MOV AH, 09H
 INT 21H
```

Or

```
.DATA
MESSAGE DB 'This is the Message to be displayed', '$'

.CODE
 LEA DX, MESSAGE
 MOV AH, 09H
 INT 21H
```

Question:

what is the importance of "$".

Does using any other register in place of DX (in function 09h) give the same result? If not, what can be the reason? Additionally, what does the LEA

TASK :

Define three strings containing your name, degree and department and display them on screen. The strings should be displayed on three different lines. Paste your code and the screenshot of output.