LAB - 1
Introduction to Assembly language

Introduction
The objective of this lab session is to introduce the student with 80X86 based microprocessors architecture, few assembly language instructions and some IO operations using INT 21H.

After this lab you will be able to

- Write simple assembly language programs.
- Assemble Link and Run assembly source code using Emu8086.

Overview:
In general, programming of a microprocessor usually takes several iterations before the right sequence of machine code instructions is written. The process, however, is facilitated using a special program called an "Assembler". The Assembler allows the user to write alphanumeric instructions, or mnemonics, called Assembly Language instructions. The Assembler, in turn, generates the desired machine instructions from the Assembly Language instructions. Assembly Language programming consists of the following steps:

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

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.

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.

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.

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)

Program 1

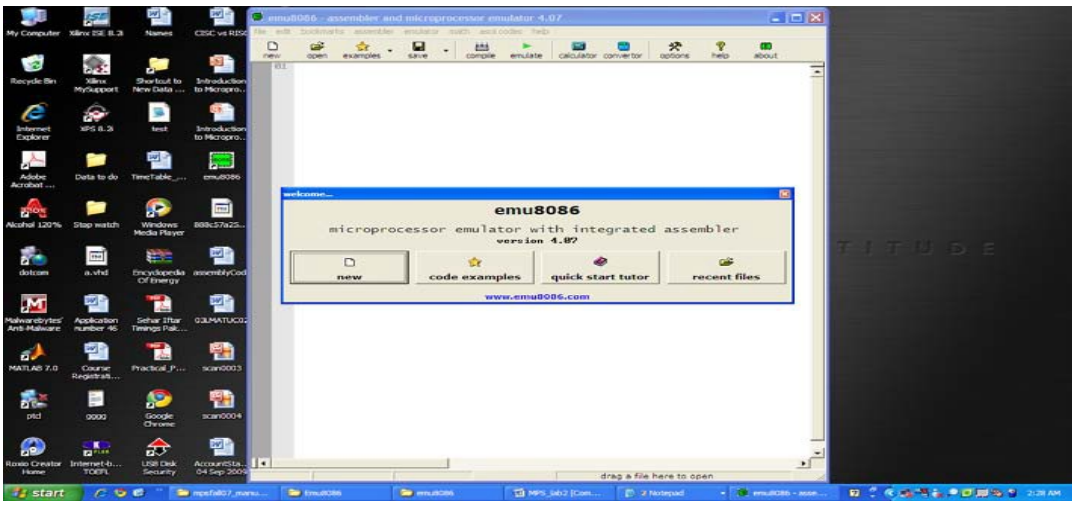
```
.MODEL SMALL              ; defines memory to be used
.STACK100H                ; defines stack of 256 bytes
.CODE                     ; code segment starts
        MOV AX, 2000      ; moves the decimal value 2000 into the AX register
        MOV BX, 2000H     ; moves the hexadecimal value 2000 into the BX register 'H' indicates a hexadecimal number
        MOV CX, 1010001B  ; moves the binary value 1010001 into the CX register 'B' indicates a binary number
        MOV DX, -4567     ; moves the decimal value -4567 into the DX register
        MOV AH,'A'        ; Copies ASCII value of"A" in AH
        MOV AL,'a'        ; Copies ASCII value of"a" in AL

        MOV AH, 4CH       ; 4CH is the function code for the DOS INT 21H interrupt that terminates a program
        INT 21H           ; INT 21h returns the control to DOS if AH=4CH
        END
```

Assembling Program in Emu8086

Emu8086 is an integrated development environment; it consists of an editor, an assembler, linker and a debugger.

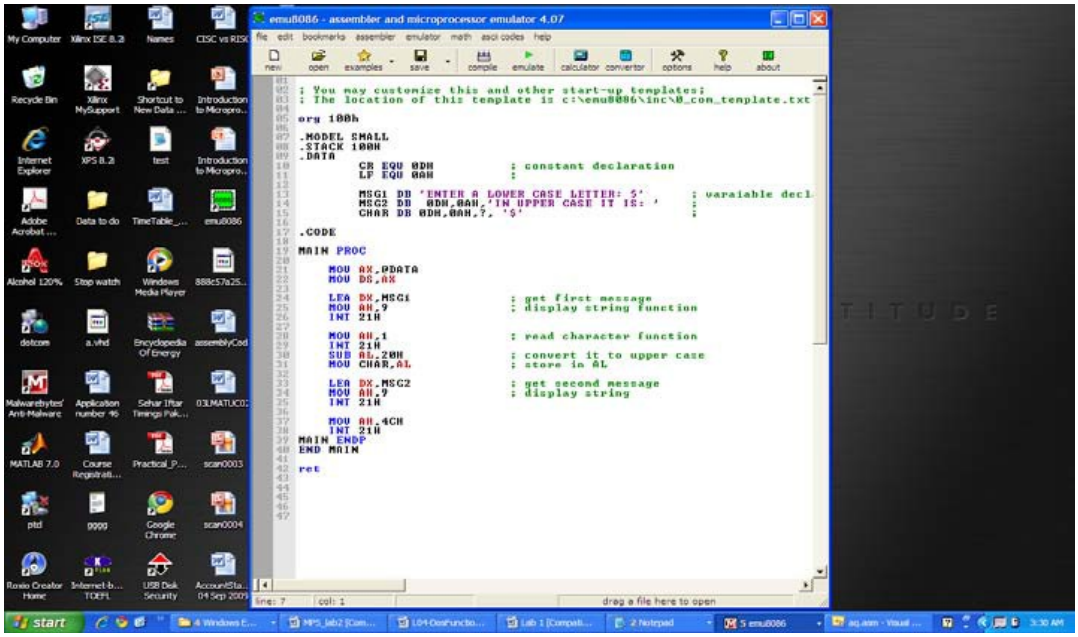
To use Emu8086 for assembling your program, do following steps.

a) Open the emu8086 present on your desktop. Following window will be appeared.



b) Select “new” from the above window. A new small window will appear select “COM template” and press Ok.

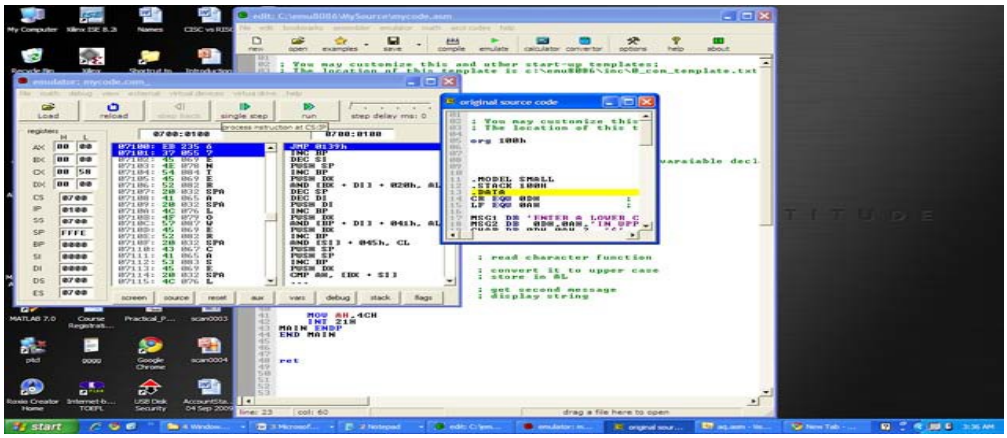
c) A code window will appear just write your specific code in the space mentioned for the code.



d) 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).

e) After saving your code, compile your code by pressing compile menu and view it there are any errors otherwise just press ok.

f) Now emulate your program form the menu and see the values of registers step by step and also get familiar with relevant things shown by the emulator.



g) Emulator screen is present for I/O.