Laboratory Exercise #1
Using the EMU8086 Integrated Development Environment

Target Course Outcome:
CO2: Designs a microprocessor-based firmware integrating a microprocessor with supporting peripherals and devices.

Objectives:
To familiarize the Emu8086 assembler/emulator to be used for executing and simulating instructions defined by the 8086-microprocessor instruction set.

Tools Required:
Emu8086 Emulator

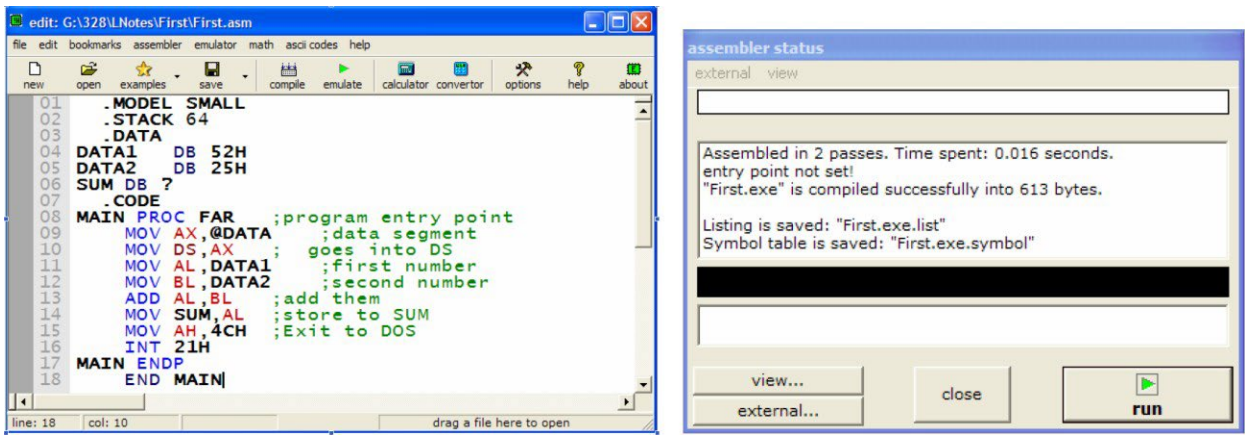
Part 1:

Emu8086 IDE

An Integrated Development Environment (IDE) provides a convenient environment to write a source file, assemble and link it to a -.COM or -.EXE file, and trace it in both source file, and machine code. Emu86 is an educational IDE for assembly program development. You can download the latest student version of EMU86 from the web page www.emu8086.com. It is a Windows program, and will run by dragging an -.ASM, -.OBJ, -.LST, -.EXE, or -.COM file into the emu86 shortcut icon. By this action, asm or lst files will start the 8086-assembler source editor, while obj and exe files starts the disassembler and debugger units.

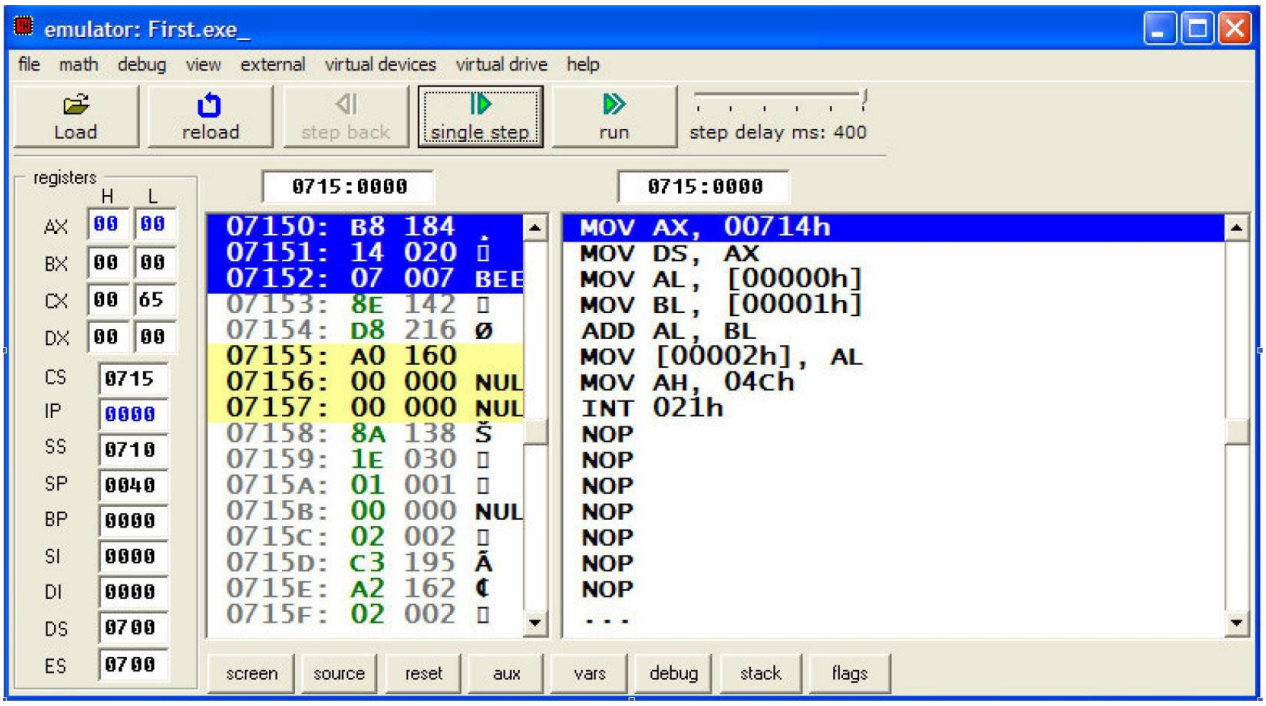
Emu8086 Source Editor

The source editor of EMU86 is a special purpose editor which identifies the 8086 mnemonics, hexadecimal numbers and labels by different colors as seen in the figure below.



EMU8086 Source Editor and assembler status report windows

The compile button on the taskbar starts assembling and linking of the source file. A report window is opened after the assembling process is completed. The figure below shows the 8086 emulator which gets opened by clicking on the emulate button.



The figure above is the Emu8086 debugging environment. It contains templates to generate command and executable files. Another benefit of Emu8086 is its emulation of a complete system, including the memory, CPU, and I/O ports, which raises opportunity to write custom programs for the different interfaces

Part II

Setting up the assembly program and using the EMU8086 assembler/emulator to compile and execute the program.

Activity #1.

Write a basic assembly language program using the MOV, INC, DEC instructions and execute the code in the Emu8086 environment.

To recall, the **MOV** instruction copies the second operand (source) to the first operand (destination). The source operand can be an immediate value, general-purpose register or memory location. The destination register can be a general-purpose register, or memory location. Both operands must be the same size, which can be a byte or a word.

The **INC** instruction is used for incrementing an operand by one. It works on a single operand that can be either in a register or in memory.

The **DEC** instruction is used for decrementing an operand by one. It works on a single operand that can be either in a register or in memory.

Procedure:

1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe
2. Create a new file an label it as Exp1.1.asm
3. In the editor window, write the following instructions.

```
ORG 100H
MOV DX, 0145H
MOV  AX, DX
INC AX
DEC DX
```

4. Press the Compile and Emulate button or press F5 key on your keyboard. The emulator window should open with this program loaded, click the Single Step button and watch the register values.

5. Write the value of the registers in the table below after each instruction as seen on the emulator window.

Instruction	Register value after the instruction is executed

Activity #2-1

Write an assembly language program using the ADD, SUB, PUSH and POP instructions and execute the code in the Emu8086 environment.

The **ADD** and **SUB** instructions are used for performing simple addition/subtraction of binary data in byte, word and double word size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.

Syntax

The ADD and SUB instructions have the following syntax:

```
ADD/SUB         destination, source
```

The ADD/SUB instruction can take place between

- Register to register
- Memory to register
- Register to memory
- Register to constant data
- Memory to constant data

However, like other instructions, memory-to-memory operations are not possible using ADD/SUB instructions. An ADD or SUB operation sets or clears the overflow and carry flags.

The PUSH/POP instructions

format:

PUSH source

POP destination

The possible operands are as follows:

source	example
register	push ax
	pop ax

The **PUSH** instruction decrements the Stack Pointer SP register (by 2) and copies a value onto the top of the stack. **POP** retrieves the value from the top of the stack and stores it into the destination, then increments the SP register (by 2).

PUSH and **POP** can be used to save and restore the values of registers when the register needs to be used for some other function temporarily or when you want to swap values between registers.

Procedure:

1. Start Emu8086 by selecting its icon from the start menu, or by running Emu8086.exe
2. Create a new file an label it as Exp1.2.asm
3. In the editor window, write the following instructions.

```
ORG  100H
MOV BX, 0123H
MOV AX, 0456H
ADD  AX,  BX
SUB AX, BX
PUSH AX
PUSH BX
```

4. Press the Compile and Emulate button or press F5 key on your keyboard. The emulator window should open with this program loaded, click the Single Step button and watch the register values.

5. Write the value of the registers in the table below after each instruction as seen on the emulator window.

Instruction	Register value after the instruction is executed

Activity #2-2

1. Write an assembly language program that will subtract the value of register 2 from register 1. Declare initial values for the 2 registers. Assume any of the general-purpose registers to be used as operand 1 and operand 2 in the subtraction process.

2. Compile and execute the program using the Emu8086 assembler/emulator.

3. Present your output.

Change log:

Date	Version	Author	Changes
Aug. 7, 2020	1.0	Rosana J. Ferolin	Initial Draft
Sep. 1, 2025	2.0	Marlowe Edgar C. Burce	Corrections in coding errors