Lab 2

Introduction to Emu8086

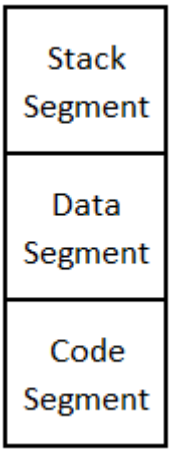
Objectives

After completing this lab

- Student will know basic structure of a program.
- Student will know about various assembler directives.
- Students will come to know how address is translated from logical to physical.
- Student will write a basic assembly language program.
- Student will learn basic rules of writing assembly instructions.
- Student will be able to debug and run their first program.

Structure of a program

Every program that is compiled using complier contains primarily two segments: Code and Data. When a program comes into execution, another segment is attached to it, called stack segment. The structure of a program during execution is show below.



Structure of assembly language program is quite similar to that of a compiled program. Data segment contains global variables and code segment contains executable code. Code segment contains main function as well as other user defined functions.

In order to define various segments in an assembly language program, assembler directives are used. Directives are commands that are part of the assembler syntax but are not related to the x86 processor instruction set. All assembler directives begin with a period (.). The following table shows various assembler directive that we will use in out program.

.model	Defines the number of code and data segments a program can have. Small: For 1 code and 1 data segment Medium: for 1 data and more than 1 code segments Large: for more than 1 code and data segments. Tiny: code and data fits in a single segment. Used for Com file.
.stack <size>	Marks the beginning of stack segment. Also define the size of stack.
.data	Marks the beginning of data segment
.code	Marks the beginning of code segment
.exit	Terminates a program.

The structure of an assembly language program is given below.

```
.model small

.stack 100h

.data
        Global variables are declared here
.code
        Code and functions are defined in this segment.

.exit
```

Registers

Registers are the storage elements inside a processor. Their size is equal to the size of a processor. Intel 8086 processor contains 16-bit register. Some register also serves for special purposes in addition to being general purpose.

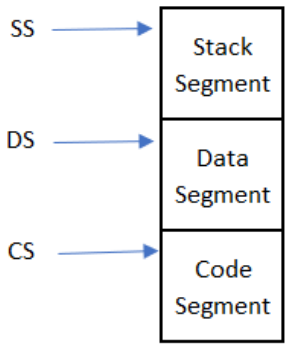
- AX - the accumulator register (divided into AH / AL).
- BX - the base address register (divided into BH / BL).
- CX - the count register (divided into CH / CL).
- DX - the data register (divided into DH / DL).
- SI - source index register.
- DI - destination index register.
- BP - base pointer.
- SP - stack pointer

Despite the name of a register, it's the programmer who determines the usage of each general-purpose register. The main purpose of a register is to keep a number. The size of the above registers is 16-bit. It's something like: 0011000000111001b (in binary form), or 12345 in decimal (human) form.

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

Besides general-purpose registers, there are some segment registers. They have very special purpose as mentioned below.

- CS - points at the segment containing the current program.
- DS - generally points at segment where variables are defined.
- ES - extra segment register, it's up to a coder to define its usage.
- SS - points at the segment containing the stack



The 8086 processor has a 20-bit address bus that can address up to 1 MB of memory. However, all the registers inside the processor are 16-bit. Therefore, a physical address cannot be stored in any register completely and hence is converted logical address containing SEGMENT: OFFSET fields. Segment and offset are both 16-bit fields. The physical address is calculated by calculating SEGMENT x 10h + offset.

Physical address for code segment is always formed using CS: IP. Once the instruction is executed, the IP is incremented by the size of instructions. In 8086, the instructions vary in sizes.

A segment is an area of memory that includes up to 64K bytes and begins on an address evenly divisible by 16 (such an address that ends in 0h).

Basic Instructions

1. MOV Destination operand, Source Operand

   mov instruction copies the contents of source operand to destination operand.

2. ADD Destination operand, Source Operand

   ADD instruction adds the contents of source and destination operands and store result back to destination operand.

3. SUB Destination operand, Source Operand

   SUB instruction subtracts the contents of source operand from the destination operand and store result back to destination operand.

Basic Rules:

- Both source and destination operands should be the same size.
- Both operands cannot be segment registers.
- The destination operand cannot be a CS or an IP register.
- If the destination operand is a segment register, the source operand cannot be an immediate value.

First assembly language program

The assembly language program

```
.model small

.stack 100h

.data

.code

Mov ax, 100h
Mov bx, 200h
Add ax, bx
Sub  ax, 25h


.exit
```

The description of the above program in shown in the following table.

Instruction	Description
Mov ax,100h	Ax ← 100h
Mov bx,200h	Bx ← 200h
Add ax, bx	Ax ← Ax + Bx
Sub ax, 25h	Ax ← Ax – 25h

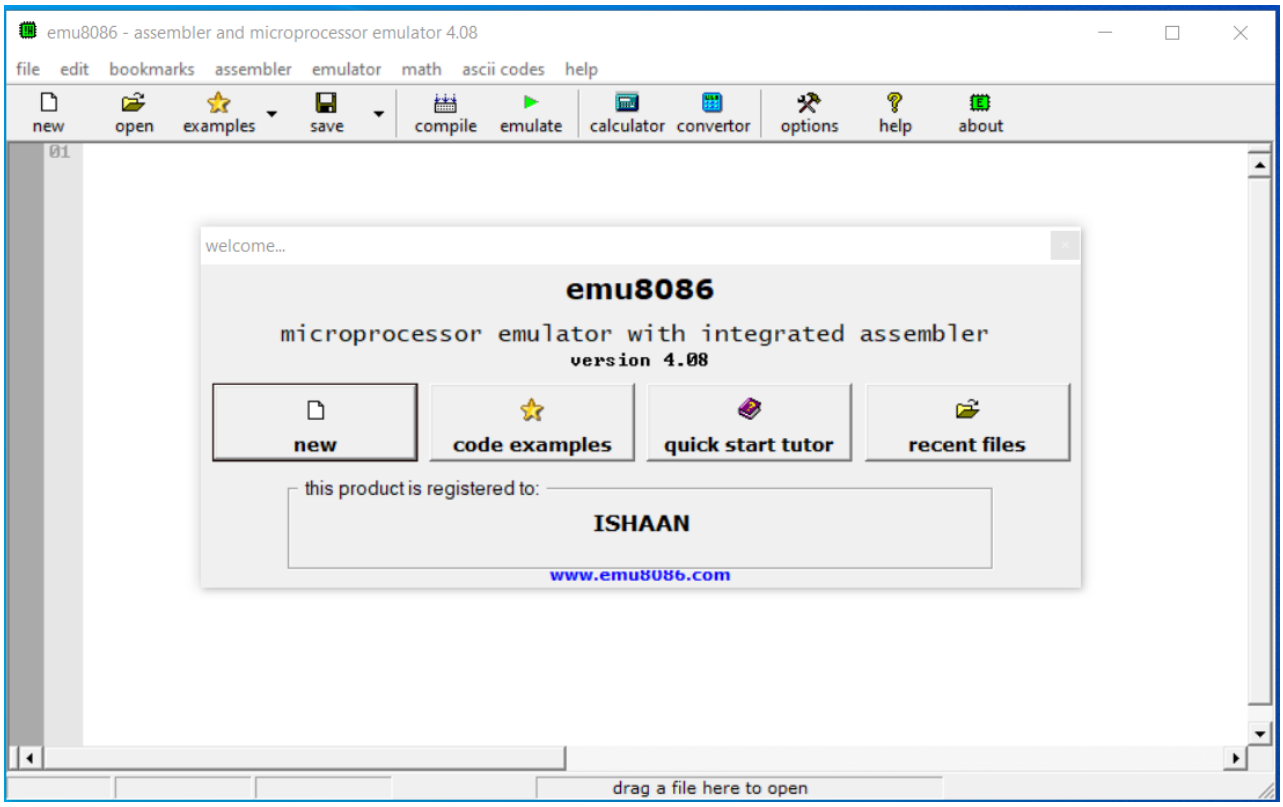
Emu8086 Tutorial Step by Step

Step-1

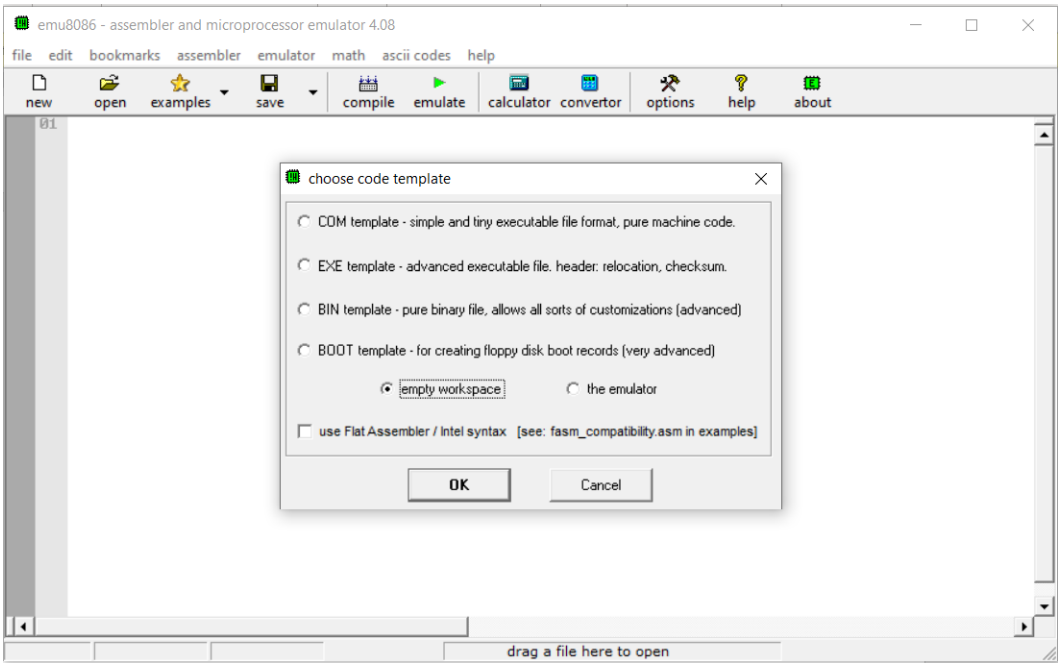
Double click on the icon on the desktop 

Step-2

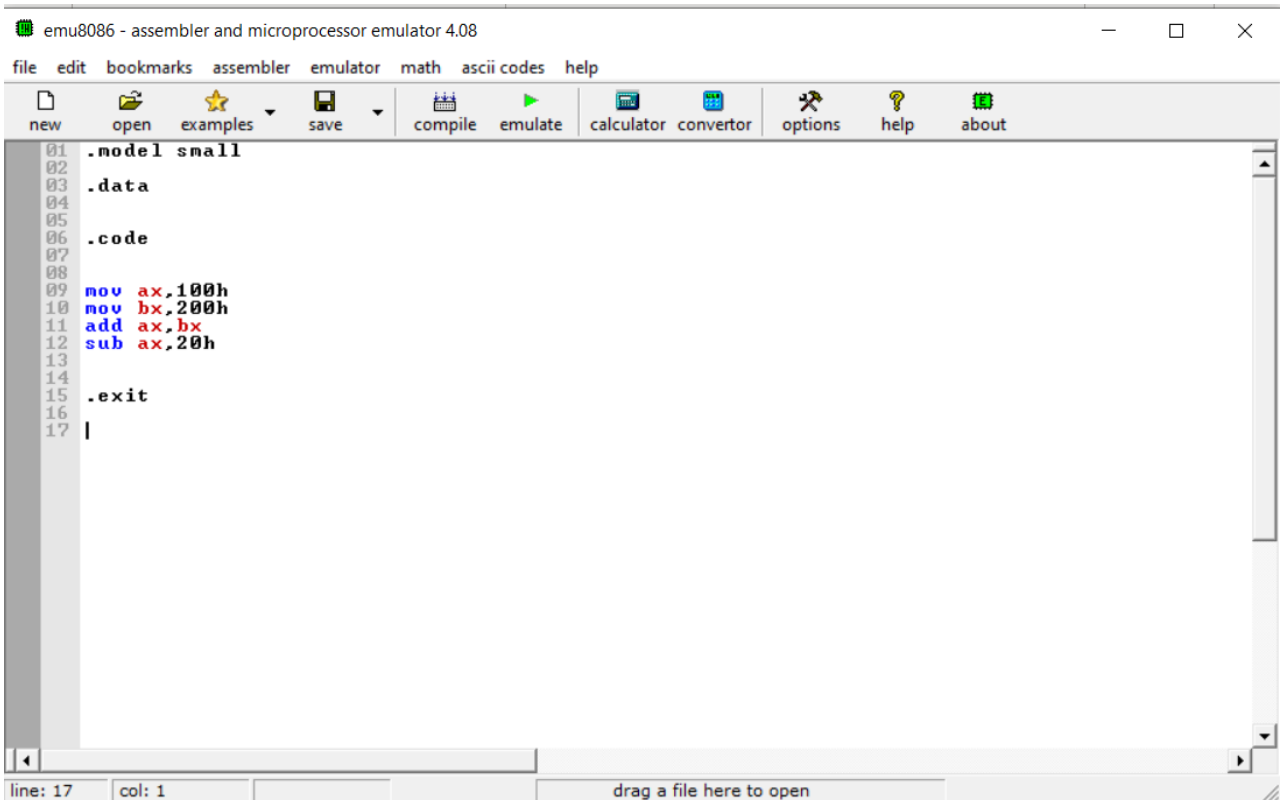
The following window will appear. Click on new.



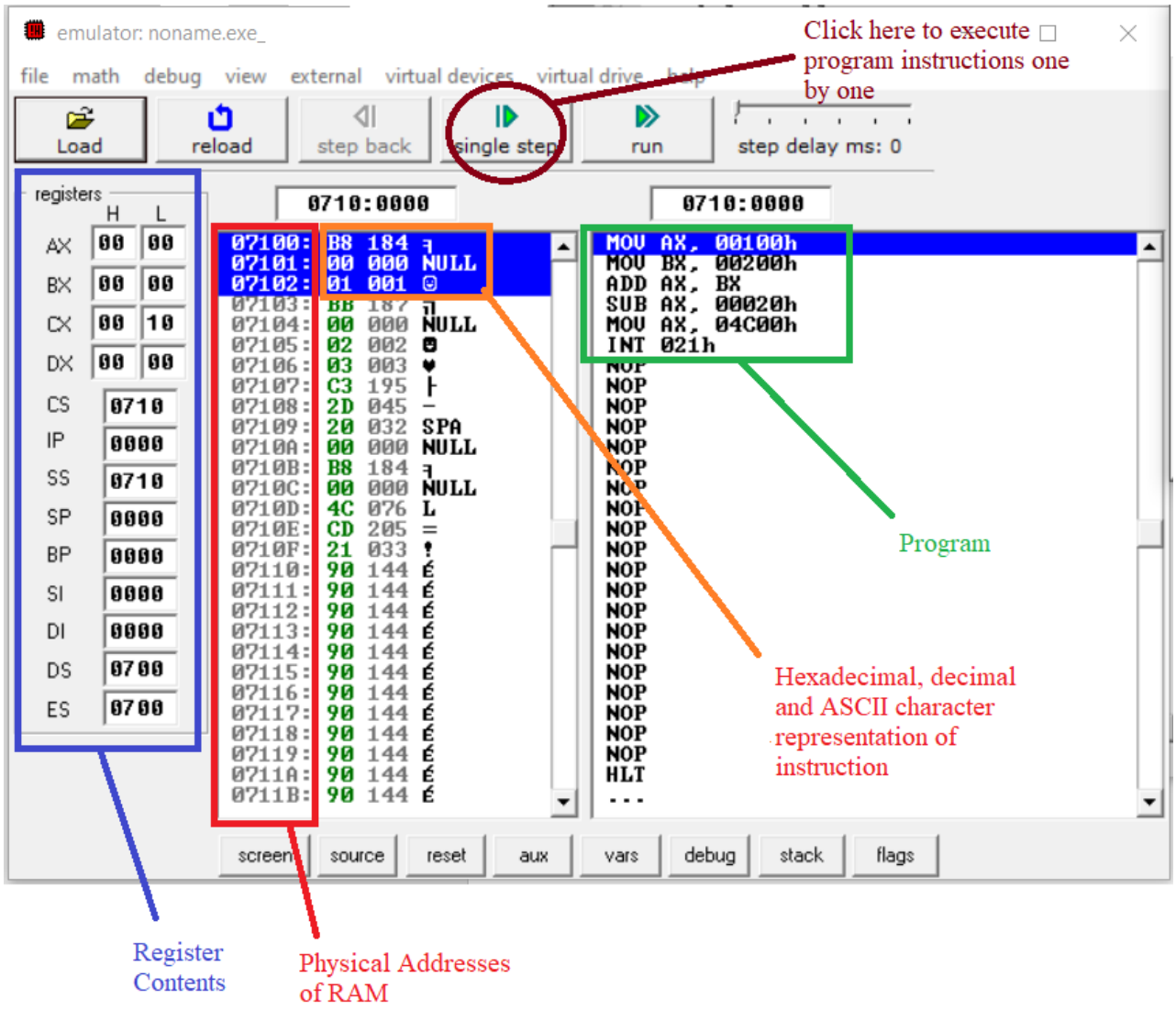
Step-3 Click on empty workspace and press OK.



Step-4 Type the code given above and clock on emulate.



Step-5 Keep clicking on "Single step" to execute program instructions one by one



Step-6 Run complete program and observe the value of various registers.

Observation:

- Observe how the physical address is being calculated. i.e., CS:IP
- Observe what number is added to the IP register after the execution of each instruction. Is that number constant? If not, why?

Practice Exercise

Task-1

Which of the following instructions are invalid. Give reasons.

a) MOV AX,27
b) MOV AL,97Fh
c) MOV SI,9516
d) MOV DS,BX
e) MOV BX,CS
f) MOV AX,23FB9h
g) MOV DS,BH
h) MOV DS,9BF2
i) MOV CS,3490
j) MOV DS,ES
k) MOV ES,BX

Task-2

Write a program in assembly language that calculates the square of six by adding six to the accumulator six times.

Task-3

Write a program to solve the following equation.

DX = AX + BH – CL + DX

Initialize the AX, BX, CX and DX registers with 0100h, 55ABh, 0A11h and 0001h values, respectively.