# CSE 331/EEE 332 (Microprocessor Interfacing & Embedded System Lab)

**Lab 01 : Introduction to registers and emu 8086**
**Lab Officer : M. A. Muhiminul Islam**

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.

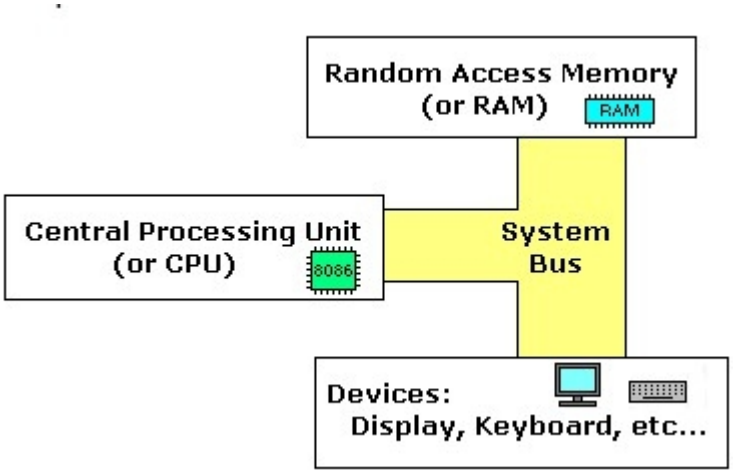
Steps required to run an assembly program:

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.

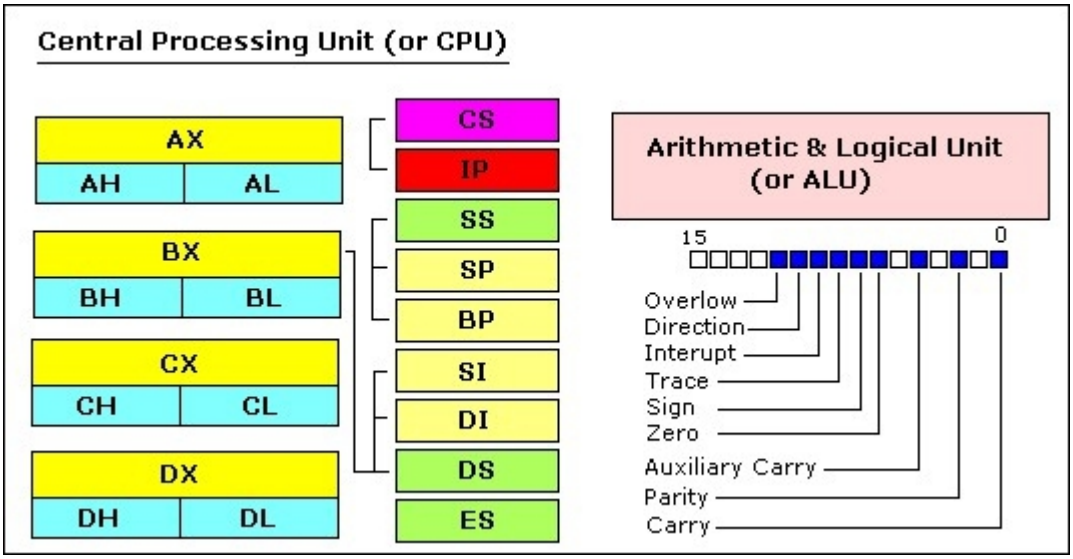
## A little about Computer Architecture

Assembly language is a low level programming language. You need to get some knowledge about computer structure in order to understand anything. Here is a simple model of a computer:



The system bus connects the various components of a computer. The CPU is the heart of the computer, most of computations occurs inside the CPU. RAM is a place to where the programs are stored in order to be executed. The CPU fetches instructions from RAM and executes it sequential order.

## 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 one in detail.

## General Purpose Registers

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 bit.

4 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 16 bit register is 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 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 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.

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 be discussed further in upcoming classes.

## Special Purpose Registers

- IP - The Instruction Pointer. Points to the next location of an 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 instructions and will serve as the basis for your first assembly program.

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

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

| Instruction | Operands | Description |
|---|---|---|
| MOV | REG, memory<br>memory, REG<br>REG, REG<br>memory, immediate<br>REG, immediate | **Copy Operand2 to Operand1.**<br><br>The MOV instruction cannot:<br>• set the value of the CS and IP registers.<br>• copy value of one segment register to another segment register (should copy to general register first).<br>• copy immediate value to segment register (should copy to general register first).<br><br>Algorithm:<br>operand1 = operand2 |
| ADD | REG, memory<br>memory, REG<br>REG, REG<br>memory, immediate<br>REG, immediate | Adds two numbers.<br><br>Algorithm:<br><br>operand1=operand1+operand2 |

## TASK 1

Write the following code in emu8086 editor:

```
org 100H

mov ax,2
mov bx,2
add ax,bx
mov cx,ax

ret
```

The first line of this program, org 100H, is a necessary requirement for all assembly programs written in emu8086. You should always start with this header. Your program should also always end with the RET instruction. This instruction basically gives back control of CPU and system resources back to the operating system. The RET statement will be used in further classes.

This program basically adds two numbers stored in two separate registers. The final result is stored in a third register. Assemble this program and run it. Follow the in class lecture regarding the use of the emulator and its various features and debugging techniques.