CSE331L – Introduction to Assembly Language

Introduction

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:

Write the necessary assembly source code

Save the assembly source code

Compile/Assemble source code to create machine code

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.

Microcontrollers vs. Microprocessors

A microprocessor is a CPU on a single chip.

If a microprocessor, its associated support circuitry, memory, and peripheral I/O components are implemented on a single chip, it is a microcontroller.

Features of 8086

8086 is a 16bit processor. It's ALU, internal registers work with 16bit binary word

8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits or 8 bits at a time

8086 has a 20bit address bus which means, it can address up to 2^20 = 1MB memory location.

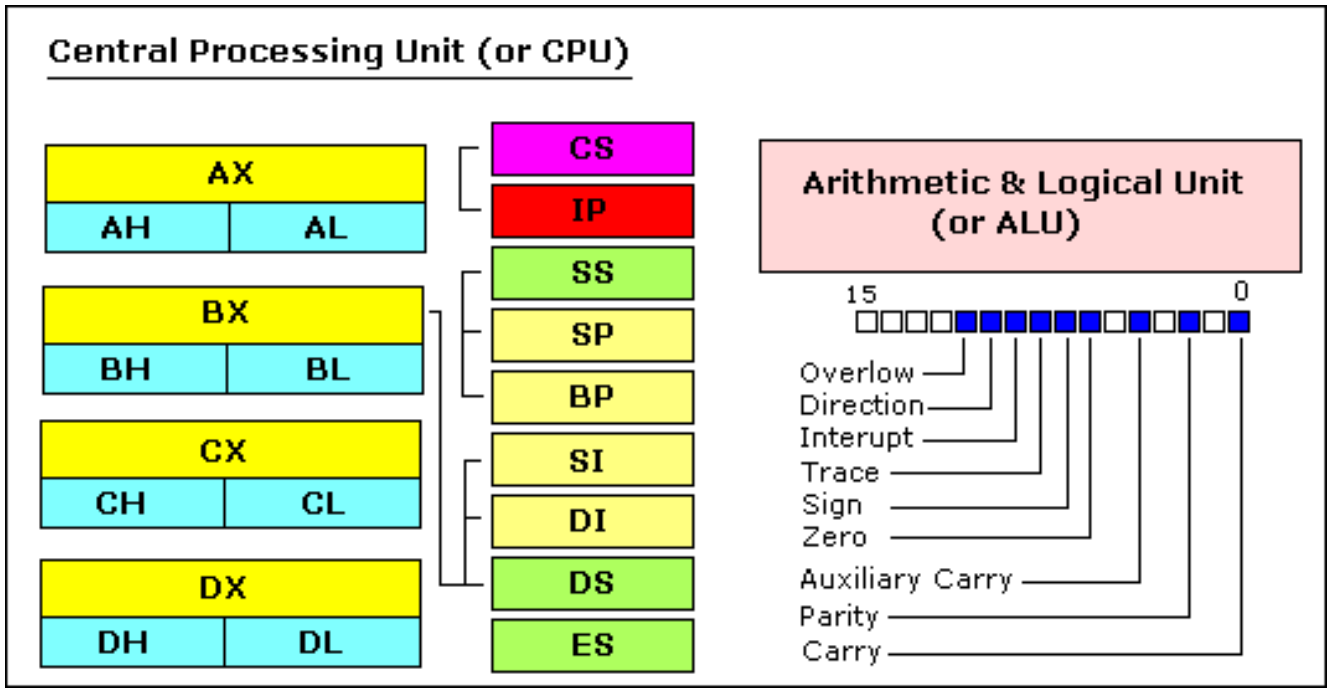
Registrar – Register – Resistor

Both ALU & FPU have a very small amount of super-fast private memory placed right next to them for their exclusive use. These are called registers

The ALU & FPU store intermediate and final results from their calculations in these registers

Processed data goes back to the data cache and then to the main memory from these registers.

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 (GPR)

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 bits.

4 general-purpose registers (AX, BX, CX, DX) are made of two separates 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 registers are 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 a 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 the 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 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 memory, REG REG, REG memory, immediate REG, immediate	Copy Operand2 to Operand1. The MOV instruction cannot: set the value of the CS and IP registers. copy value of one segment register to another segment register (should copy to general register first). copy an immediate value to segment register (should copy to general register first). Algorithm: operand1 = operand2
ADD	REG, memory memory, REG REG, REG memory, immediate REG, immediate	Adds two numbers. Algorithm: 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.

" ORG (abbr. for ORiGin) is an assembly directive (not an instruction). It defines where the machine code (translated assembly program) is to place in memory. As for ORG 100H this deals with 80X86 COM program format (COMMAND) which consist of only one segment of max. 64k bytes. 100H says that the machine code starts from address (offset) 100h in this segment, effective address is CS:100H. For com format the offset is always 100H. I suppose that addresses 0 to 100H could be used by bios, but I am not that sure. Another example is ORG 7C00H for intel exe program format."

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.