COURSE: **CMPS 250 – Machine Organization and Assembly Language Programming**

(Prerequisite: CMPS 144) An introductory study of the organization and architecture of computers undertaken through an exploration of various virtual machines. Programming at the assembly language level and interfacing with software components (primarily written in Java and C). Topics include representation of data and instructions, instruction sets, addressing modes, computer arithmetic, memory hierarchies, digital logic, microprogramming, pipelining, and parallel processing. (Essentially from Undergraduate Catalog 2007-2008)

Machine Organization – For historical reasons it is named "machine organization", but could just as well be termed "computer organization", because the machine in question is a computer. Either way, we mean to take a look at the underlying components of computer systems. Consider the analogy of learning to drive a car and then one day you open the hood and begin a study of the internal workings of the machine you have been using. If you have been programming in one or more "high-level" languages (such as Java, C++, C, etc.) then CMPS 250 provides an opportunity to "pop the hood" and begin understanding how what you have programmed actually happens.

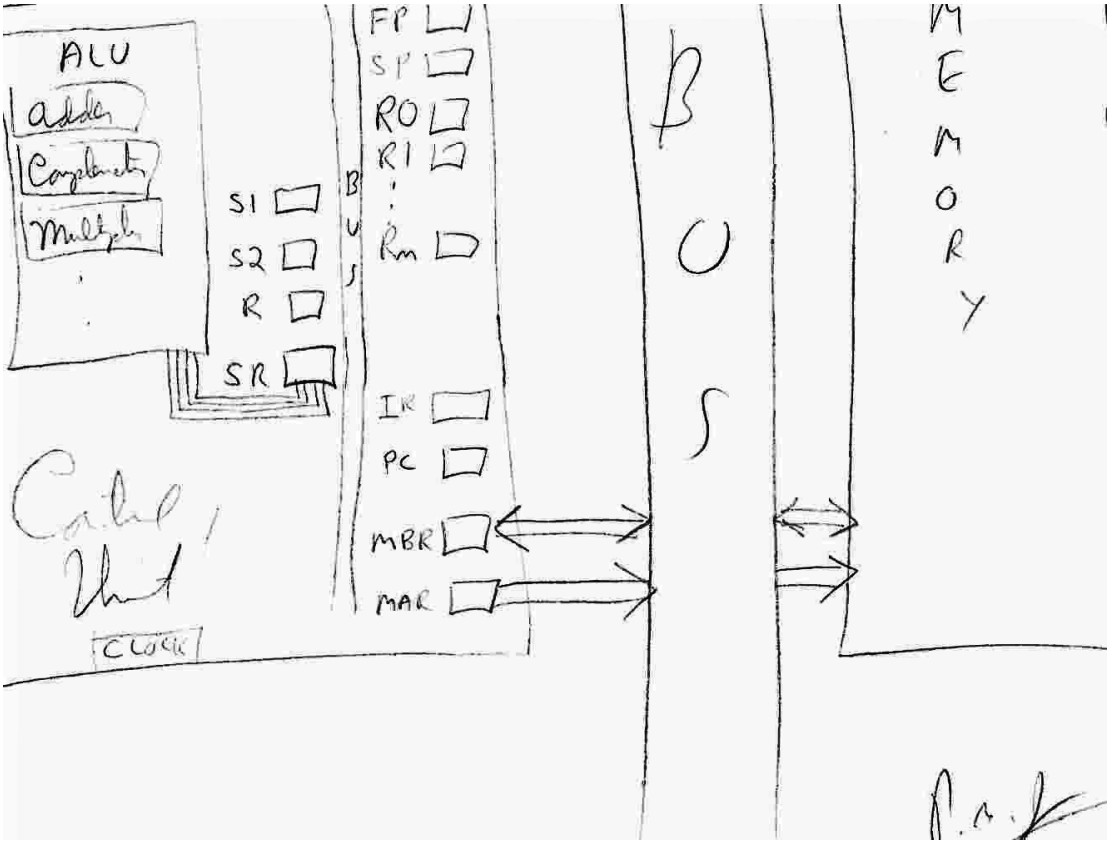
One of the clearest statement on the distinction between "organization" and "architecture" is presented in the text Computer Organization & Architecture, Designing for Performance – 7th Edition, by William Stallings, Prentice Hall, 2006. On page 7 he states,

> "Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program."
>
> "Computer organization refers to the operational units and their interconnections that realize the architectural specifications."

For the sake of clarification, a system may be defined as "any collection of components defined to work together for some purpose".

Using this definition, a computer system may be depicted by this crude diagram that identifies the relevant hardware components:

- Processor (also showing ALU, Control Unit, Internal Bus, and Registers)
- External Bus
- Memory (i.e. RAM)



Although Software is not explicitly depicted its presence and role is essential to understand the system, and the roles of each component.

Corresponding and potentially clearer diagrams appear in our course text, The Essentials of Computer Organization and Architecture – Second Edition, by Linda Null and Julia Lobur, Jones and Bartlett, 2006. Specifically Figure 1.4 on page 32.

High-level languages – Most programming languages fall into this category; including Java, C++, C, PHP, Pyton, etc. Such languages normally require a translator (termed a compiler) to translate valid source programs into equivalent programs expressed in lower level languages. Often, but not always high-level language programs are translated into the machine language of the underlying hardware architecture.

Machine Language – In some sense, machine language is the lowest level language used to express programs. Each processor has its own unique machine language, and such languages are typically numeric (i.e. binary) making them readily able to be processed. Machine languages are thus cryptic and exist mostly to suit the needs of the underlying hardware processors.

Assembly Language – Is regarded as a "low-level" language since it has a very close relationship to its corresponding "machine language". It is most accurate to characterize assembly language as "symbolic machine language", meaning that instead of representing operations and operands as numbers (which is the case with machine language), these are represented by mnemonics and identifiers. It is necessary to have a translator to convert "assembly language source programs" into equivalent "machine language programs", and such a translator is termed an "assembler" as it is far simpler than a compiler.