Computer Science 220
Assembly Language & Comp. Architecture
Siena College
Fall 2010

Topic Notes: Introduction and Overview

Welcome to Assembly Language & Computer Architecture!

Why take this course?

To learn about how computers are designed and built, right from the low-level details of the electrical circuits up to the assembly language code generated by compilers, and to see how that assembly language can be used to perform the common operations required by high-level language programs.

You have seen the idea of an abstract data type.

This idea of *abstraction* is key in this course as well. There are many levels of abstraction that take us from electronic circuits to the kinds of programs we write.

Level 0: Physics and Digital Logic

- properties of atoms, electrons, and quantum dynamics
- semiconductors/silicon used to build transistors
- transistors used to build logic gates (CMOS)
- Boolean logic
- combinational logic, arithmetic circuits
- sequential logic, finite state machines
- architectural issues: *i.e.,* caches, virtual memory, pipelining

Level 1: Conventional Machine Language

- CPUs, vonNeumann architecture
- most of the processors we think about operate at this level
- we will consider MIPS primarily and x86 (IA-32) to a lesser degree – other modern examples include PowerPC, Sparc, ARM
- programmed in "machine code" – just a bunch of 0's and 1's
- the *instruction set architecture* is defined here: what are capabilities of the machine?

Level 2: Assembly Language

- human readable/writable description, closely related to machine language

- translated to machine language by an *assembler*

Level 3: Operating System

- runs programs on the hardware
- manages sharing of resources
- *e.g.*, Linux, FreeBSD, Mac OS X, Android, Windows, etc.

Level 4: Conventional Programming Langauages

- at this level, we implement applications software – the programs people actually use
- *compilers* translate this code into assembly, which the assembler in turn translates to machine code to run on the computer
- make use of services provided by the operating system
- *e.g.*, C, C++, Fortran

Not everything fits perfectly into this particular abstraction hierarchy. We might also consider:

Level 3.5: Virtual Machines

- an abstract "machine" that runs programs in a higher-level "architecture"
- the virtual machine is in turn implemented to execute in the traditional environment
- *e.g.*, JVM, .NET

Level 4.5: Interpreted Languages

Java, BASIC, Python, Perl

To understand what's going on inside the computer, you need to understand all of these levels. Even if you are sure that you will never work at the lower levels, understanding these levels is important. It will make you a better high-level language programmer.

Our focus is on the lowest levels.

- Digital logic
- Microarchitecture and instruction set architecture
- Assembly language programming
- Higher-level computer architecture topics (parallelism, caches, virtual memory)