Computer Systems & Programming (CS 367)

Prof. Sanjeev Setia
George Mason University

Goal of the course
Understand how computers work -- bridge the gap between high level programming in Java/C/C++ and machine level programming

Approach:
Build understanding from the bottom up.

Bits \ Gates \ Processor \ Instructions \ C Programming

You will be able to write programs in C and understand what’s going on underneath. You will also get some experience with “systems programming”
Relationship to other courses

Prerequisites
- CS 211 (Computer Science II)
- ECE 301/303/331 (Digital Logic)

CS 112, 211, 310 have migrated to using Java instead of C++

CS 265 (Assembly programming) no longer taught

Goal of this course (CS 367) is to replace CS 265 (Assembly Programming) and expose students to a different programming language (C) from a bottom up perspective

Programming assignments involving machine and assembly language (LC3), and programming in C.

Course Textbook

Introduction to Computing Systems: From Bits and Gates to C and Beyond
2nd Edition
Yale N. Patt
Sanjay J. Patel
Logistics

Grading
• Four Programming Assignments (40%)
  ➢ to be completed individually
  ➢ programs will be tested on a Linux platform for grading
    – You need to obtain an IT&E Unix Cluster account (See syllabus)
• Midterm & Final (50%)
• Two homework (written) assignments (10%)
• Several extra credit opportunities

Class TA
• Kishore Vemulpali (svemulpa@gmu.edu)
  • Office Hrs: TBA, Room 365 (S&T II)
• My office hrs
  • M 1:30-2:30 pm, Room 430 (S&T II)

Course Outline

Bits and Bytes
• How do we represent information using electrical signals?
• Will not be discussed at length since you should already know this

Digital Logic
• How do we build circuits to process information?
• covered in ECE 301/303/331

Processor and Instruction Set
• How do we build a processor out of logic elements? Covered in CS 365
• What operations (instructions) will we implement?

Assembly Language Programming
• How do we use processor instructions to implement algorithms?
• How do we write modular, reusable code? (subroutines)

I/O, Traps, and Interrupts
• How does processor communicate with outside world?

C Programming
• How do we write programs in C? First two weeks of semester
• How do we implement high-level programming constructs?

Systems Programming
• How do assemblers, linkers, and loaders work?
• How to write programs that use services provided by the operating system?
Two Recurring Themes

Abstraction

- Productivity enhancer – don’t need to worry about details…
  Can drive a car without knowing how the internal combustion engine works.
- …until something goes wrong!
  Where’s the dipstick? What’s a spark plug?
- Important to understand the components and how they work together.

Hardware vs. Software

- It’s not either/or – both are components of a computer system.
- Even if you specialize in one, you should understand capabilities and limitations of both.

Big Idea: Universal Computing Device

All computers, given enough time and memory, are capable of computing exactly the same things.

PDA = Workstation = Supercomputer
**Turing Machine**

Mathematical model of a device that can perform any computation – Alan Turing (1937)
- ability to read/write symbols on an infinite “tape”
- state transitions, based on current state and symbol

Every computation can be performed by some Turing machine. *Turing’s thesis*

\[
\begin{align*}
\text{T}_{\text{add}} & \quad \text{a,b} \rightarrow a+b \\
\text{T}_{\text{mul}} & \quad \text{a,b} \rightarrow ab
\end{align*}
\]

Turing machine that adds  
Turing machine that multiplies

For more info about Turing machines, see [http://www.wikipedia.org/wiki/Turing_machine/](http://www.wikipedia.org/wiki/Turing_machine/)

For more about Alan Turing, see [http://www.turing.org.uk/turing/](http://www.turing.org.uk/turing/)

---

**Universal Turing Machine**

A machine that can implement all Turing machines – this is also a Turing machine!
- inputs: data, plus a description of computation (other TMs)

\[
\begin{align*}
\text{T}_{\text{add}} & \quad \text{T}_{\text{mul}} \rightarrow \text{U} \\
\text{a,b,c} & \rightarrow c(a+b)
\end{align*}
\]

Universal Turing Machine

\text{U} is *programmable* – so is a computer!
- instructions are part of the input data
- a computer can emulate a Universal Turing Machine

* A computer is a universal computing device.
From Theory to Practice

In theory, computer can *compute anything* that’s possible to compute
  • given enough *memory* and *time*

In practice, *solving problems involves* computing under constraints.
  • time
    ➢ weather forecast, next frame of animation, ...
  • cost
    ➢ cell phone, automotive engine controller, ...
  • power
    ➢ cell phone, handheld video game, ...