About Computer Organization

CS 350: Computer Organization & Assembler Language Programming

A. Why?

- Course guidelines are important.
- Before studying how computers are organized, we should think about what they do.
- We should also think about any fundamental differences between computers.
- The programming environment we'll be using (C under Linux) is important to know.

B. Outcomes

At the end of today, you should:

- Know how the course will be structured and graded.
- Know that computation involves manipulation of symbols.
- Know that (up to memory and speed constraints) all computers can do the same computations.
- Know that we'll be programming using C on `alpha.cs.iit.edu`, a Linux machine.

C. Introduction and Welcome

- The course webpages are at http://www.cs.iit.edu/~cs350
- The home page includes news and the calendar of lectures, coursework, and tests.
- There's a Course Plan page that discusses the textbook, the course topics, course structure, and grading, collaboration, disability, and ethics policies.

D. What Are Computers?

- So what do computers do, anyway? [See Activity 1.1]
- One important use of computers involves manipulating data represented by symbols and numbers (or symbols that represent numbers).

- In some sense, computers don't actually manipulate symbols, they move photons and electrons etc.
- We think of them as manipulating symbols or doing calculations because of our interpretation of these activities.
- I.e., we view things from some context and abstract away the parts that aren't part of the context.
- Optional textbook identifies levels of abstraction and transformations in computer systems: Problem, Algorithm, Language (Program), Machine (Instruction Set Architecture), Micro-architecture, Circuits, Devices.
- Compiler translates high-level program into machine code (a sequence of bits).
- Machine runs machine code that comes from a particular set of instructions (the ISA: Instruction Set Architecture). The ISA is the interface between software and hardware.
- The ISA instructions themselves can be broken down into sub-operations; this is the micro-architecture of the machine. The micro-operations and -operands are implemented using logic and arithmetic circuits (which deal with bits). The circuits are implemented using devices (such as transistors, switches) that fiddle with electrons.

E. Is There An Ultimate Computer?

- Obviously some computers are more powerful than others, assuming we measure power as speed or memory etc.
- If we abstract away from speed and memory constraints, is any computer more powerful than another?
- Turns out they are equivalent in power in the sense that they all do the same kinds of calculations: Given enough time and memory, any computer can simulate another computer.
- E.g., the "Virtual PC" program ran on Power PC Macintoshes and made them simulate Intel hardware, so you could run Windows (very slowly :-).

F. Turing Machines

- Alan Turing was a mathematician & cryptologist; he worked on the British Enigma project that broke Nazi codes and made them readable by the Allies.
- (Link to Alan Turing Wikipedia article.)
- He invented what we call "Turing Machines" (TMs).
- Very simple theoretical computer.
- Has a tape with a read/write head.
- You have instructions like "If we're in state 72 and we see a $, move the head right one symbol and go to state 53."
- Unsurprisingly, you can simulate a TM with a modern computer.
- Surprisingly, TMs can simulate modern computers (verrrry slowly).
- You can design TMs that do arithmetic on integers and floating-point numbers, manipulate characters, and so on.
- You could even simulate an Intel PC.
- There even exists a "Universal" TM.
- The Universal TM is an interpreter/simulator of TM programs: You give it a description of the other TM and the other TM's data, and the universal machine will do what the other TM would've done.
- In some sense, this is the only TM you would ever need to actually build.
- In the same way, we don't generally build different computers to (e.g.) edit documents, edit photographs, send/receive email, etc.
- We typically build general computers and then write programs to make them simulate the computations some more-specialized machine might do.

G. Turing's Thesis and Universal Computation Devices

- Ignoring speed/memory constraints, all computers are equivalent. Rough argument:
    1. Take two computers X and Y.
    2. For each, write a program that simulates the universal TM.
    3. Write a universal TM program that simulates X and another for Y.

- For step 3 to work, you have to assume that any computation that can be done by some computer can be done by some TM. This is Turing's Thesis:
- **Turing's Thesis**: Every computation that can be done can be done by a TM.
- Another way to say this is that TMs are Universal Computation Devices: They can do every conceivable kind of mechanical computation.
- Turing's thesis is not formally provable, but it sure looks to be true: Nobody's found a mechanical, computational operation that a TM can't do[1].
- If Turing's thesis is true, then modern computers are also universal computation devices because TMs and modern computers are equivalent (can solve the same set of problems).

H. Feasible Computations

- Computers can do every known mechanical computation, but not necessarily in a reasonable amount of time. There is a large set of problems that are all equivalent in the time it takes to solve them, but that time is infeasible for large inputs.
- In CS 430 you'll study problems with feasible solutions (and analyze the algorithms to see how fast they are). In CS 530 you'll study the larger question of what problems are known to have no possible mechanical solution.

I. Our Programming Environment

- We'll be writing programs in C and compiling and running them on `alpha.cs.iit.edu`, a Linux machine.
- You're welcome to do your development on whatever system you have (Windows, Mac OS, or Linux) but for correctness tests, we'll use alpha for testing. We'll compile using `gcc -Wall -std=c99 -lm` *filename.c*.
- `gcc` is the GNU C compiler; the `-Wall` option says to print all compiler warnings, `-std=c99` says to compile using the C99 syntax for C, and `-lm` says to include the math library (see Lab 1 for more discussion).

---

1 Some people think that quantum computers will violate Turing's Thesis.

About Computer Organization

CS 350: Computer Organization & Assembler Language Programming

Note: Activities are not handed in or graded

A. Why?

- Before studying how computers are organized, it might be good to think about what they do!

B. Outcomes

After this activity, you should know something about

- The common features there are in the ways we use computers.
- The ways (or lack of ways) that computers can be extended to perform calculations that can't otherwise be performed.

C. Problems

1 Give a number of examples of kinds of data that computers manipulate. (Yeah, it's a vague problem.) It may help to think of the kinds of datatypes found in programming languages.

2. Are there any fundamental similarities or differences between your answers to Problem 1?

3. You're familiar with the kinds of operations found in programming languages like Java. Can you think of any operations that can be added to computers that can't be implemented (assuming memory, running time, and other resources aren't issues)?

4. (Trick question) Where/when do you turn in the answers to these problems?