Slide 1

Administrivia

- The bookstore claims to have lots of copies of the textbook. If they're not on the shelves, I'd suggest asking an employee.
- First homework to be on Web later today. Due next Friday.

Slide 2

"It's All Ones and Zeros"

- Computers only "understand" binary. Why? Easier to design hardware in terms of two distinct states (rather than, say, 10). Hence everything — data and instructions — must be represented in terms of 0 and 1.
- How to represent data?
  - Integers — binary numbers, as in CS 1320.
  - Non-integers (e.g., `double`) — "floating point" (later).
  - Characters — small binary numbers (ASCII, Unicode).
  - MS Word files, MP3s, etc., — more complex, but similar ideas.
- How to represent instructions? "Machine language". Very briefly for now:
  - Processor has a small repertoire of operations, each with limited number of operands.
  - Each "instruction" is an operation plus operands. Can represent it using a number for the instruction and numbers for the operands.

Slide 3

A Short History of Programming

- In the beginning — "programming" by physically rewiring hardware or setting switches.

  Arguably better than no computer at all, but tedious and slow!

- First advance — represent program as a long sequence of bits and store in memory ("stored program" concept).

  Now "programming" means putting together that sequence of bits and loading it into memory. Better, but still tedious and error-prone.

Slide 4

A Short History of Programming, Continued

- Next advance — express programs symbolically, but still in terms of the processor's operations — "assembly language".

  So we "program" by writing in symbolic notation and translating to binary by hand. Still tedious, still error-prone, but less so.

- Next advance — let the machine do the translating.

  So we program as above, but "assembler" program translates. Still not easy, but a lot less tedious and error-prone.

Slide 5

A Short History of Programming, Continued

- Next advance — represent program more abstractly, make the machine do more work.

  So we program in a "high-level language" and use "compiler" program to translate.

  Much less tedious, slow, and error-prone for programmers!

  Also, programs can be "portable".

  Resulting code is efficient if compiler is good — initially not always true, but now usually the case.

- Next/associated advance — develop code to be used by many applications — "library" routines and operating systems.

  Again, makes programmer's job less tedious, etc.,

- Nice picture on p. 7.

Slide 6

Hardware Components — An Abstract View

- Input devices — way to get info into computer from outside. Examples include mouse and . . . ?
- Output devices — way to get information back to outside world . Examples include display and . . . ?
- Processor — "brain" that does actual calculations, etc. Can divide into
  - "Datapath" — stores values, performs operations (e.g., addition).
  - "Control" — "puppet master" for datapath.
- Memory — stores values, "scratch pad" for calculations.
- The above are the "classic five". Other noteworthy components (really I/O devices):
  - Storage devices (e.g., disk).
  - Network interfaces.

Slide 7

"Layers of Abstraction" Idea

- Idea of "layers of abstraction" used over and over in CS.
- In software, you know how this works.

  Example — "shopping cart" abstraction, implemented using "resizable array" abstraction, implemented using "linked list" abstraction . . .

  Goal — "manage complexity" by dividing big complicated problem into manageable parts.

Slide 8

"Layers of Abstraction" Idea, Continued

- Same idea can be used in hardware design, for the same reason. So we talk about
  - "Instruction set architecture" (ISA or architecture) — a definition/specification of how the hardware behaves, detailed enough for programming at assembly-language level.

    E.g, "x86 architecture", "MIPS architecture", "IBM 360 architecture".
  - "Implementations of an architecture" — actual hardware that behaves as defined. Can have many implementations of an architecture, allowing the same program executable to run on (somewhat) different hardware systems.

    E.g., Intel chips, SGI chips, IBM 360 family of processors.

Slide 9

A Little About Integrated Circuits

- Conceptual view of hardware:
  - "Transistor" — on/off switch controlled by electrical current.
  - Combine/connect a lot of transistors to get "circuit" that does interesting things (e.g., addition).
  - Put a bunch of circuits together to get a "chip" / "integrated circuit" (IC). If lots of transistors, "VLSI chip".

Slide 10

A Little About Integrated Circuits, Continued

- Manufacturing process starts with a thin flat piece of silicon, adds metal and other stuff to make wires, insulators, transistors, etc.
- Of course, this is all automated! Low-level chip designers use CAD-type tools, which save designs in a standard format, which the chip designers simulate/test with other software, and then send off to be "fabricated".
- Typically make many "chips" on a "wafer", discard those with defects, bond each good one to something larger with "pins" to allow connections to other parts of computer.

Slide 11

Minute Essay

- We said a program as stored in memory is a sequence of ones and zeros. Name three ways a programmer could produce this sequence. (Hint: some ways involve the use of other programs.)