Slide 1

Administrivia

- Midterm grade summaries e-mailed. Most people doing well; exceptions are people not turning in work consistently.
- Reminder: Homework 6 due today.
- Homework 7 on the Web. Due in two weeks.
- (Yes, I did change the topic for today. Sorry about the last-minute switch but it makes more sense to put off the originally-planned topic to a time closer to when you'll need it for homework.)

Slide 2

Homework 5 Essays

- Several people found the problems difficult (particularly the second one — the first some found easy). But several also found them interesting or good practice.
- One person started by saying "Is it weird that I enjoyed this assigment?" I say no; it just means you like programming?
- Another said of the second problem "I don't think I have ever been more happy to get a program to work." We've all been there?
- (Sample solution posted if you're curious about my solution to the second problem. Several people tried the suggested approach, with varying degrees of success.)

Slide 3

Data Representation — "It's All Ones and Zeros"

- At the hardware level, all data is represented in binary form — ones and zeros. (Why? hardware for that is apparently simpler to build.)
- How then do we represent various kinds of data? First a short review of binary numbers . . .

Slide 4

Binary Numbers (Review?)

- Humans usually use the decimal (base 10) number system, but other (positive integer) bases work too. (Well, maybe not base 1.)
- In base 10, there are ten possible digits, with values 0 through 9.

  In base 2, there are 2 possible digits ("bits"), with values 0 and 1.
- Everything in base 2 works the same as base 10, if you think about how base 10 actually works, so to speak:

  In any base, digits represent increasing powers of the base (so, 1s, 10s, 100s, 1000s, etc., for base 10, and 1s, 2s, 4s, 8s, etc., for base 2).

Slide 5

Converting Between Bases (Review?)

- Converting from another base to base 10 is easy if tedious (just use definition).
- Converting from base 10 to another base? Two algorithms for that ...

Slide 6

Decimal to Binary, Take 1 (Review?)

- One way is to first find the highest power of 2 smaller than or equal to the number, write that down, subtract it from the number, and continue.
- In pseudocode (letting n be the number we want to convert):

while $(n > 0)$
  find largest p such that $2^p \leq n$
  write a 1 in the p-th output position
  subtract 2^p from n
end while

(A little sloppy in that we don't explicitly say that other output positions are 0.)

Slide 7

Decimal to Binary, Take 2 (Review?)

- Another way produces the answer from right to left rather than left to right, repeatedly dividing by 2 (again n is the number we want to convert):

  while $(n > 0)$

  divide n by 2, giving quotient q and remainder r

  write down r

  set n equal to q

  end while

Slide 8

Octal and Hexadecimal Numbers (Review?)

- Binary numbers are convenient for computer hardware, but cumbersome for humans to write. Octal (base 8) and hexadecimal (base 16) are more compact, and conversions between these bases and binary are straightforward.
- To convert binary to octal, group bits in groups of three (right to left), and convert each group to one octal digit using the same rules as for converting to decimal (base 10).

  (Why this works: Write out definition, factor a power of 8 out of each group of 3 digits.)
- Converting binary to hexadecimal is similar, but with groups of four bits. What to do with values greater than 9? represent using letters A through F (upper or lower case).

Slide 9

Computer Representation of Integers (Review?)

- So now you can probably guess how non-negative integers can be represented using ones and zeros — number in binary. Fixed size (so we can only represent a limited range).
- How about negative numbers, though? No way to directly represent plus/minus. Various schemes are possible. The one most used now is *two's complement*: Motivated by the idea that it would be nice if the way we add numbers didn't depend on their sign. So first let's talk about addition . . .

Slide 10

Machine Arithmetic — Integer Addition and Negative Numbers (Review?)

- Adding binary numbers works just like adding base-10 numbers — work from right to left, carry as needed. (Example.)
- Two's complement representation of negative numbers is chosen so that we easily get 0 when we add $-n$ and n.

  Computing $-n$ is easy with a simple trick: If m is the number of bits we're using, addition is in effect modulo 2^m. So $-n$ is equivalent to $2^m - n$, which we can compute as $((2^m - 1) - n) + 1)$.
- So now we can easily (?) do subtraction too — to compute $a - b$, compute $-b$ and add. (This simplifies one part of processor design — more in Computer Design!)

Slide 11

Binary Fractions

- We talked about integer binary numbers. How would we represent fractions?
- With base-10 numbers, the digits after the decimal point represent negative powers of 10. Same idea works in binary.

Slide 12

Computer Representation of Real Numbers

- How are non-integer numbers represented? usually as *floating point*.
- Idea is similar to scientific notation — represent number as a binary fraction multiplied by a power of 2:

$$x = (-1)^{sign} \times (1 + frac) \times 2^{bias+exp}$$

and then store $sign$ $frac$, and exp. Sign is one bit; number of bits for the other two fields varies — e.g., for usual single-precision, 8 bits for exponent and 23 for fraction. Bias is chosen to allow roughly equal numbers of positive and negative exponents.

- Current most common format — "IEEE 754". Read up on it sometime (Wikipedia article seems okay) — *lots* of "who knew?" details!

Slide 13

Numbers in Math Versus Numbers in Programming

- The integers and real numbers of the idealized world of math have some properties not completely shared by their computer representations.
- Math integers can be any size; computer integers can't.
- Math real numbers can be any size and precision; floating-point numbers can't. This has some interesting consequences . . .

  (Two "floating point is strange" examples.)

Slide 14

Real Numbers in Math Versus Floating Point

- Some values that can be represented easily in decimal can't be represented in binary floating-point.
- Math operations on reals have properties such as associativity that don't necessarily hold for floating point (!).

Slide 15

Computer Representation of Text

- We talked already about how "text strings" are, in C, arrays of "characters". How are characters represented? Various encodings possible.
- One common one is ASCII — strictly speaking, 7 bits, so fits nicely in smallest addressable unit of storage on most current systems (8-bit byte).
- Another one is Unicode — originally 16 bits (Java's `char` type), now more complicated. (Again, Wikipedia article seems okay.)
- Either encoding can be considered as "small integers".
- C's `char` type often ASCII but doesn't have to be. (Older systems use(d) EBCDIC, an encoding rooted in punched cards.) C also has `wchar_t`, which *could* be Unicode.

Slide 16

Minute Essay

- TBA