Slide 1

Administrivia

- Reminder: Homework 7 due next Wednesday.

  One more homework. (So we're close to being done!)

Slide 2

Minute Essay From Last Lecture

- Most people had seen BSTs — apparently they're just now being covered in CS2 — so that's good. If you're one of the exceptions, the homework writeup has some suggestions, or please come talk to me.

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 simpler to build.)
- How then do we represent various kinds of data? First a short review of binary numbers . . .

Slide 3

Binary Numbers

- 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.

Slide 4

Slide 5

Computer Representation of Integers

- 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 doesn't depend on their sign. So first let's talk about addition ...

Slide 6

Machine Arithmetic — Integer Addition and Negative Numbers

- 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.

Slide 7

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 8

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:

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

  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 9

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. Also, some quantities that can be represented easily in decimal can't be represented in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)
- (Two "floating point is strange" examples.)

Slide 10

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.

Minute Essay

- I don't have a definite plan for the next two classes, but some things we could look at are multithreading (OpenMP and/or Pthreads) or text-mode full-screen processing with `ncurses`. Or — other requests?

Slide 11