Slide 1

Administrivia

- (None?)

Slide 2

Sidebar: `bc`, Some Advice About Text-Mode Programs

- Last time I showed using the simple command-line calculator `bc`? a little more about it . . .
- From its `man` page, it's "an arbitrary precision calculator language". Simplest usage is what I showed in class — type arithmetic expressions and have them evaluated. Also supports variables. Full details in `man` page.
- How to exit? like many text-mode Linux programs, control-D ("end of file") works. Most of them also have a keyword to exit — "quit", "exit", etc.

  Another way to exit — and this will work for your C programs too — is control-C, which interrupts the program. Not graceful but usually works.

Slide 3

Number Representation, Revisited

- As discussed previously, most digital computers use fixed-size binary numbers to represent non-negative integers.
- What about negative integers? several schemes have been tried. Most common now is "two's complement" (no, I don't like the apostrophe either). Before defining, talk a little about addition . . .

Slide 4

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 5

Machine Arithmetic — Integer Multiplication

- Multiplying binary numbers also works just like multiplying base-10 numbers — for each digit of the second operand, compute a partial result, and add them.
- (This can get slightly tricky, when adding more than two partial results involves carrying, but basic idea is straightforward extrapolation from how it works in base 10.)

Slide 6

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 7

Computer Representation of Real Numbers

- How are non-integer numbers represented? usually as *floating point*. "IEEE 754 standard" spells out details; most current hardware implements it.
- 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.

Slide 8

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 exactly in binary.
- Math operations on integers and reals have properties such as associativity that don't necessarily hold for the computer representations. (Yes, really!)

Slide 9

C and Representing Numbers — Integers

- Computer hardware typically represents integers as a fixed number of binary digits, using "two's complement" idea to allow for representing negative numbers.
- C, like many (but not all!) programming languages bases its notion of integer data on this, but also has a notion of different types with different sizes.
- Unlike many more-recent languages, C defines for each type a minimum range rather than a definite size. Intent is to allow efficient implementation on a wide range of platforms, but means some care must be taken if you want portability.

Slide 10

C and Representing Numbers — Real Numbers

- Hardware also typically supports "floating-point" numbers, with a representation based on a base-2 version of scientific notation. This allows representing not only fractional quantities but also allows representing larger numbers than would be possible with fixed-length integers. Notice that only fractions that can be written with a denominator that's a power of two can be represented exactly.
- Again C goes along with this and provides different "sizes" (`float` and `double`). As with integers, exact sizes not specified, only minimum criteria.

Slide 11

Text Data

- Remember that computers represent everything using ones and zeros. How do we then get text? well, we have to come up with some way of "encoding" text characters as fixed-length sequences of ones and zeros — i.e., as small(ish) numbers.
- (To be continued later in the semester.)

Slide 12

Minute Essay

- Any questions about today's material, or number representation in general?