Slide 1

Administrivia

- Homework 1 grades and comments to be sent via e-mail. This is how you'll get feedback on your assignments.
- Most people seemed not-flummoxed by `vim`. Keep in mind as you use this tool that if you don't learn more than the absolute minimum you're likely to find it painful to use. Consider periodically reviewing "how to" tips. You can find mine on the course Web site, under "useful links".

Slide 2

Minute Essay From Last Lecture

- Almost everyone got a right answer to the first question, and many for the second. A good start! This topic isn't the most important we'll talk about, but I think it's useful background.

Slide 3

Binary Numbers — Recap

- Binary (base 2) in some sense works just like decimal (base 10), if you think about how base 10 works — digits represent powers of the base, starting with 1 and increasing as you move left.
- Converting from base 2 to base 10 is easy if tedious. The other way is a little harder; we looked at two methods last time.

Slide 4

Octal and Hexadecimal Numbers

- 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).

  (Think for a few minutes about why this works.)
- 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).
- (Examples.)

Slide 5

Computer Representation of Integers

- Computers represent everything in terms of ones and zeros. For non-negative integers, you can probably guess how this works — number in binary. Fixed size (so we can only represent a finite 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

- Adding binary numbers works just like adding base-10 numbers — work from right to left, carry as needed. (Example.)
- But what if the number is too big to represent? "Overflow". What happens then varies depending on hardware. One possibility is to simply discard high-order bits that don't fit, so for an m-bit integer addition is in effect modulo 2^m. Another is to signal a hardware error.

Slide 7

Machine Arithmetic — Negative Numbers

- Two's complement representation of negative numbers is chosen so that we easily get 0 when we add $-n$ and n, assuming addition "wraps around" (i.e., addition is module 2^m where m is the number of bits).

  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 means that in designing a processor to do arithmetic, once you make something that can add, you get subtraction almost "for free".)

Slide 8

Machine Arithmetic — Integer Multiplication and Division

- Simple hardware to multiply and divide basically follows the procedures humans can do on paper — multiply by computing and adding "partial sums", divide via long division.
- Details can get a little tricky, but basic idea is straightforward extrapolation from how it works in base 10.
- (Probably worth noting at this point that the simple methods for doing all four arithmetic operations are not especially fast, so many processors use clever tricks to speed them up. Beyond the scope of this course!)

Slide 9

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.

  (Example.)

Slide 10

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 11

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! We'll look at an example later.)

Slide 12

Binary Numbers — Recap

- General review of binary numbers and arithmetic on them is meant as background for understanding how computers represent integers internally. That in turn is meant to help you understand some of the limitations associated with C integer types.
- Discussion of how to convert among number systems is mostly I-think-useful background, but also because the two ways of converting from decimal to binary show how there can be more than one way to solve a problem.

Slide 13

Minute Essay

- The IEEE 754 standard defines one way of splitting up the bits of a floating-point number into a field for the significant digits and a field for the exponent. But a different standard might split them differently. What would happen if more bits were used for the significant digits? What would happen if more bits were used for the exponent?

Slide 14

Minute Essay Answer

- More bits for the significant digits means you can represent numbers with more precision, but the range of magnitudes is smaller. More bits for the exponent means a larger range of magnitudes, but less precision.