Slide 1

Administrivia

- Reminder: Homework 2 due Thursday.
- Reminder: Quiz 2 Thursday. Likely topics are programs from class — “what does this program print?” or “write a program to do this task”.

Slide 2

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 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 3

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 4

Machine Arithmetic — Bit Shifting

- With base-10 numbers, multiplying (and dividing) by powers of 10 is easy, right? just shift the decimal point.
- Same idea applies to binary numbers and powers of two — "bit shifting".

Machine Arithmetic — Integer Multiplication

Slide 5

- 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 tricky, when adding more than two partial results involves carrying.)

Binary Fractions

Slide 6

- 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*.
- 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 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

Type Conversions

- Implicit conversions: When you assign a value of one type to another (e.g., `float` to `int`), or write an expression that mixes types, C will perform an implicit conversion.
- Explicit conversions: Putting a type in parentheses before an expression means you want to convert to the indicated type. Example:

  ```
  (double) (1 / 2)
  ```

  versus

  ```
  (double) 1 / (double) 2
  ```

  This is called *casting*.

Slide 10

Minute Essay

- What is the base-10 value of the binary number $.101_2$? (Okay to write as a fraction or even as an expression.)
- What are you finding interesting, annoying, difficult, etc., about Homework 2?

Slide 11

Minute Essay Answer

- $.101_2$ is $2^{-1} + 2^{-3}$, i.e., $5/8$.