CSC161 2010F *Imperative Problem Solving*

Class 13: IEEE Floating Point Representation

Held: Friday, 17 September 2010

Summary: We consider issues in the representation of real numbers, focusing on the design decisions in the IEEE floating-point representation.

Related Pages:

- EBoard.
- Reading: Stone & Rebelsky: IEEE Floating Point Representation of Real Numbers..

Notes:

- Assignment 4 is now ready.
- What did you think about the experience of doing assignment 3?

Overview:

- C's bitwise operations.
- Religious wars: Big-endian vs. Little endian representation>.
- The problem of real numbers.
- One approach: Rationals.
- Another approach: Fixed-precision.
- Detour: Scientific notation.
- The IEEE floating point standard.

Bitwise Operations in C

Logical

- & - bitwise "and"
  - 0 and 0 is 0
  - 0 and 1 is 0
  - 1 and 0 is 0
  - 1 and 1 is 1
- | - bitwise "or"
  - 0 or 0 is 0
  - 0 or 1 is 1
  - 1 or 0 is 1
  - 1 or 1 is 1
- ~ - bitwise "not"
  - not 0 is 1

    - not 1 is 0
    - Why is this different than negate?
- We can use these to extract bits from an integer.
    - To access the kth bit of i, compute 2^k and and it with i
    - If the result is 0, the bit was 0. If the result is non-zero (true, in C), the bit was 1.
- We can use these to change bits in an integers
    - To change the kth bit of i, compute 2^k and or it with i.
- We often use integers to store a variety of flags (settings)
    - One bit per flag
    - If the bit is on, the flag is set
    - If the bit is off, the flag is not set

Shifting

- << - left shift
- >> - right shift
- Lots of variants.
- Typically used for faster multiplication and for other clever operations.

<

- If an integer is four bytes, how to we number the bytes?
- Two normal answers, both with extreme proponents.

The Problem of Real Numbers

Representing Reals

- We've found ways to represent integers. Will that help with real numbers?
- One approach, similar to that for characters: Map between real numbers and integers.
    - Unfortunately, there isn't a natural mapping of reals onto the integers.
    - See your Math professor for more details.
- If we're going to used a fixed number of bits, we're unlikely to be able to represent many irrational numbers or particularly small numbers, so we will never be able to get all the reals in a particular range. We must approximate them.
- We'll design a few different representations and I'll challenge you to critique them.
    - You are allowed to reflect on the reading in your criticisms.

One approach: Rational numbers

- Since we have no irrational reals, we can represent many real numbers fairly easily as rational numbers: ratios of two integers.
- Reserve some number of bits for the numerator, represented as a signed integer in the notation of your choice.

- Reserve some number of bits for the denominator.

Another approach: Fixed precision

- A common early representation of reals was the *fixed-point* strategy.
- Given a sequence of bits to use to represent a real number, select a position for the "decimal" point (except that it should be called the "binary" point).
- The column to its left is the 1's column, the next column to the left is the 2's, then the 4's, and so on and so foth.
- The column to the right of the point is the 1/2's column. The next column is the 1/4's, and so on and so forth.
- Note that this is effectively a representation of rational numbers with a fixed denominator.

Detour: Scientific Notation

- Rather than trying to design a representation from scratch, perhaps we should reflect on common practices in other disciplines.
- Many folks who work with real numbers (including approximations of real numbers) use *scientific notation*, as in "1.231×10^5".
- What is involved in scientific notation?
  - A *sign*, which indicates whether the number is positive or negative,
  - A *mantissa*, which gives the primary digits of the number.
  - A *base of exponentiation* (in this case, 10).
  - An *exponent*.
- Might we not use the same general technique in designing a representation for real numbers?
- Such notations are called *floating point* because the point moves depending on the exponent.

IEE Single-Precision Floating-Point Numbers

- Rather than having each computer manufacturer decide on a particular floating point representation, standards groups worked do design a few standard representations.
- Two of the most popular representations are IEEE Single-Precision and Double-Precision floating-point numbers.
  - We'll concentrate on single-precision numbers. Double-precision numbers follow similar conventions.
- Single-precision numbers use 32 bits.
  - One bit gives the sign.
  - Eight bits give the exponent (represented in bias-127 notation).
  - The remaining 23 bits give the mantissa, using fixed-point notation.
- Note that the base of exponentiation is not represented. It is instead fixed at 2.
- There are some important restrictions on the various parts.
  - The exponent must be between -126 and 127
  - The mantissa must be no smaller than 1.0 and cannot be as large as 2.0.
- Note the clever trick: Given that restriction on the mantissa, you don't need to represent the first bit

of the mantissa (since it's always 1).

Special Cases

- There are some parts the previous discussion ignores.
- The range of exponents only gives 254 different values. We have two additional unused values, corresponding to bit sequences 00000000 and 11111111.
- If the mantissa can be no smaller than 1.0, how do we represent 0?

Special Case: Representing Small Values

- If the exponent consists of only 0's, we use -126 as the exponent and restrict the mantissa to values no less than 0 and less than 1.
- Hence, 0 has an exponent of all 0's and a mantissa of all 0's.

Special Case: Error Values

- If the exponent consists of only 1's, the float represents some special value.
- Positive infinity uses all 0's for the mantissa, as does negative infinity (sign bit gives the sign).
- Anything else is NaN.

Lab

- Lab.
- This is an optional lab. We probably won't have time for it.
