March 08

- Chapter 4 – Floating Point
- The second exam will be 22 March, the SECOND Class after Spring Break.
- NO CLASS on 20 March. Use that time to STUDY! The TAs will hold office hours during that time.

What is the problem?

- Many numeric applications require numbers over a VERY large range. (e.g. nanoseconds to centuries)
- Most scientific applications require fractions (e.g. ?)

But so far we only have integers.

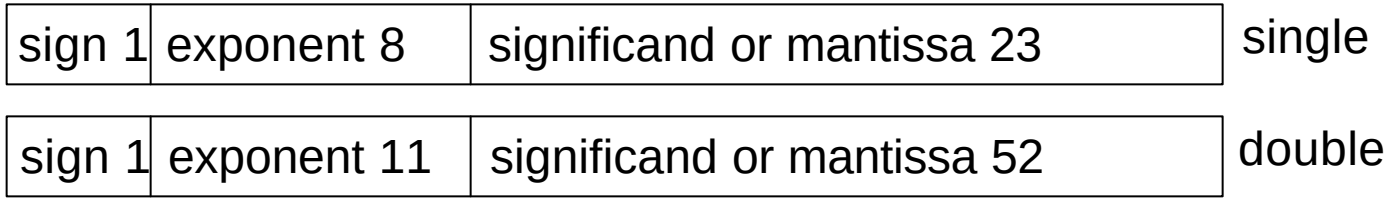
We *COULD* implement the fractions explicitly (e.g. ½, 1023/102934)

We *COULD* use bigger integers

Floating point is a better answer for most applications.

Floating Point

- Just ***Scientific Notation*** for computers (e.g. -1.345 ? ?? ? ?? ?
- Representation in IEEE 754 floating point standard



  - value = (0-sign) ? significand ? $2^{exponent}$
  - more bits for significand gives more precision
  - more bits for exponent increases range

Remember, these are JUST BITS. It is up to the instruction to interpret them.

Normalization

- In Scientific Notation:
  $1234000 = 1234 \times 10^3 = 1.234 \times 10^6$
- Likewise in Floating Point
  $1011000 = 1011 \times 2^3 = 1.011 \times 2^6$ ← Normalized
- The standard says we should always keep them normalized so that the first digit is 1 and the binary point comes immediately after.
- But wait! There's more! If we know the first bit is 1 why keep it?

IEEE 754 floating-point standard

- Leading “1” bit of significand is implicit
- We want both positive and negative exponents for big and small numbers.
- Exponent is “biased” to make comparison easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: (0-sign) ????significand) ??$2^{\text{exponent – bias}}$
- Example:
  - decimal: -.75 = -3/4 = $-3/2^2$
  - binary: -.11 = -1.1×2^{-1}
  - floating point: exponent = 126 = 01111110
  - IEEE single precision: 10111111000000000000000000000
- What about zero?

Arithmetic in Floating Point

- In Scientific Notation we learned that to add to numbers you must first get a common exponent:
  - 1.23 x 10^3 + 4.56 x 10^6 ==
  - 0.00123 x 10^6 + 4.56 x 10^6 ==
  - 4.56123 x 10^6
- In Scientific Notation, we can multiply numbers by multiplying the significands and adding the exponents
  - 1.23 x 10^3 x 4.56 x 10^6 ==
  - (1.23 x 4.56) x 10^(3+6) ==
  - 5.609 x 10^9
- We use exactly these same rules in Floating point PLUS we add a step at the end to keep the result normalized.

Floating point AIN'T NATURAL

- It is CRUCIAL for computer scientists to know that Floating Point arithmetic is NOT the arithmetic you learned since childhood

- 1.0 is NOT EQUAL to 10*0.1 (Why?)
  - 0.1 decimal is a NON TERMINATING FRACTION in binary!
  - Lots of other numbers are too.
  - So how useful is the == comparison?

- Floating Point arithmetic IS NOT associative
  - x + (y + z) is not necessarily equal to (x + y) + z

- Addition may not even result in a change
  - (x + 1) MAY = x

Floating Point Complexities

- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divide by zero yields “not a number”
- Implementing the standard can be tricky
- Not using the standard can be even worse

Chapter Four Summary

- Computer arithmetic is constrained by limited precision
- Bit patterns have no inherent meaning but standards do exist
  - two's complement
  - IEEE 754 floating point
- Computer instructions determine "meaning" of the bit patterns
- Performance and accuracy are important so there are many complexities in real machines (i.e., algorithms and implementation).
- Accurate numerical computing requires methods quite different from those of the math you learned in grade school.