Floating Point (a brief look)

- We need a way to represent:
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .00000001
  - very large numbers, e.g., 3.15576 x 10^9

- Representation:
  - sign, exponent, significand: \((-1)^{\text{sign}} \times \text{significand} \times 2^{\text{exponent}}\)
  - more bits for significand gives more accuracy
  - more bits for exponent increases range

- IEEE 754 floating point standard:
  - single precision: 8 bit exponent, 23 bit significand
  - double precision: 11 bit exponent, 52 bit significand

- Leading “1” bit of significand is implicit
- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: \((-1)^{\text{sign}} \times (1 + \text{significand}) \times 2^{\text{exponent} - \text{bias}}\)

- Example:
  - decimal: -.75 = -3/4 = -3/2^2
  - binary: -.11 = -1.1 x 2^-1
  - IEEE single precision: 10111111010000000000000000000000

Floating Point Complexities

- Operations are somewhat more complicated (see text)
- 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”
  - other complexities
  - Implementing the standard can be tricky
  - Not using the standard can be even worse
    - see text for description of 80x86 and Pentium bug!

Floating Point Add/Sub

- To add/sub two numbers
  - We first compare the two exponents
  - Select the higher of the two as the exponent of result
  - Add the significand portions
  - Results will be 1.xxxxx… or 0.1xxxx….
  - In the first case shift result right and adjust exponent
  - Round off the result
  - This may require another normalization step

Floating Point Multiply

- To multiply two numbers
  - Add the two exponent (remember access 127 notation)
  - Produce the result sign as exor of two signs
  - Multiple significand portions
  - Results will be 1.xxxxx… or 01.xxxxx….
  - In the first case shift result right and adjust exponent
  - Round off the result
  - This may require another normalization step

Floating Point Divide

- To divide two numbers
  - Subtract divisor’s exponent from the dividend’s exponent
    (remember access 127 notation)
  - Produce the result sign as exor of two signs
  - Divide dividend’s significand by divisor’s significand portions
  - Results will be 1.xxxxx… or 0.1xxxx….
  - In the second case shift result left and adjust exponent
  - Round off the result
  - This may require another normalization step