Class 4: Representing Information: Floating Point

- Floating Point Representation
- Floating Point Arithmetic
- IEEE Floating Point Formats
- ASCII

Simple Floating Point Format

- Scientific notation:

  $n = f \times 10^{e}$, where
  - f is the fraction, or mantissa and
  - e is a positive or negative integer called the exponent
- The computer representation version of this is called *floating point*
- Examples:

  $32.67 = 3.267 \times 10^{1}$

  $-0.25 = -2.5 \times 10^{-1}$

Floating Point Addition and Subtraction

- First, adjust the values so the exponents are the same.
- Then, add or subtract the mantissas.
- Limited precision floating point may require numbers to be rounded or truncated in order to fit into the number of bits available for the mantissa, resulting in a loss of accuracy.

Floating Point Multiplication and Division

- If multiplying, add the exponents and multiply the mantissas.
- If dividing, subtract the exponents and divide the mantissas.
- Example:

  1.2e3 * 2.0e2 = 2.4e5 (240000)

  1.2e3 / 2.0e2 = 0.6e1 (6)

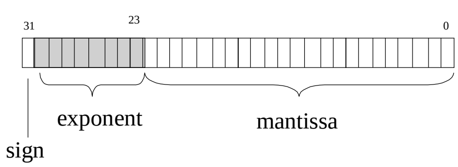
Comparing Floating Point Numbers

- There are inaccuracies present in any computation.
- This makes comparisons very dangerous.
- Testing for equality is not a good idea.
- If absolutely necessary, find an error that you will allow (a tolerance) and check to see if the two values are within the error range, rather than absolutely equal.

IEEE Floating Point Formats

- Three floating point formats:
  - 32 bit single precision
    - 1 sign bit
    - 8 exponent bits
    - 24 mantissa bits
  - 64 bit double precision
    - 1 sign bit
    - 11 exponent bits
    - 53 mantissa bits
  - 80 bit bit extended precision
    - 1 sign bit
    - 15 exponent bits
    - 64 mantissa bits
- Notice: for single and double precision numbers the total is one bit too many! Why?

Single Precision Floating Point



- Sign – one bit 0 for positive, 1 for negative
- Exponent – eight bit, excess-127 format (add 127 to actual exponent value)
- Mantissa – 23 bit sign magnitude (sign bit gives sign). 24th (high order) bit is always one and not stored.

Normalization

- To get maximum precision, computations use *normalized* values.
- A normalized floating point value is one where the higher order mantissa bit is equal to one.
- This is done by shifting the bits to the left and decrementing the exponent for each shift until the left-most bit is one.
- So how can you store a 24 bit mantissa in 23 bits?
  - If the left-most bit is always one, then you don't need to store it!

Normalization, cont.

- So how does normalization work?
- Each shift left is the equivalent of multiplying by two.
- Decrementing the exponent is the equivalent of dividing by two.
- Example:
  - 0011e4 = $3 * 2^4 = 48$
  - 0110e3 = $6 * 2^3 = 48$
  - 1100e2 = $12 * 2^2 = 48$
- Since the left most bit is always one in a normalized number, you can save a bit of storage by not storing it.

Converting to Single Precision

- Example:
  - 27.4 decimal
- First, convert to binary
  - 27^{10} = 0001 1011
  - .4 = ?
  - .4 * 2 = 0.8 → 0, .8 left
  - .8 * 2 = 1.6 -> 1, .6 left
  - .6 * 2 = 1.2 -> 1, .2 left
  - .2 * 2 = 0.4 -> 0, .4 left
  - .4 * 2 = 0.8 – this will repeat!
  - so,
  - $27.4 = 11011.\overline{0110} * 2^0$

Converting to Single Precision, cont.

- Next, need to normalize:
  - $27.4 = 00011011.\overline{0110} * 2^0$
  - $= 1.1011\overline{0110} * 2^4$
- Compute exponent, extend to eight bits if necessary (not needed in this case):
  - exponent = $4_{10} + 127_{10} = 131_{10}$
  - $131_{10} = 10000011_2$
- Shift mantissa by one bit (since higher order one bit is implied) and extend the repeating portion for the appropriate number of bits (23)
  - 10110110011001100110011
- Result:
  - 0 10000011 1011011 0011001100110011

Another example

- 16.2 decimal

Converting from Single Precision into Decimal Floating Point

- Example:
  BD500000h
- First, convert to binary:
  BD500000h = 1011 1101 0101 0000 0000 0000 0000 0000
- Then, pull out the various components:
  - Sign – negative
  - Exponent – 01111010
  - Mantissa – 101 0000 0000 0000 0000 0000

Converting from Single Precision into Decimal Floating Point

- Convert the exponent:
  01111010 = 64 + 32 + 16 + 8 + 2 = 122 (excess-127)
  122 – 127 = -5
  exponent = -5
- Convert the mantissa:
  101 0000 0000 0000 0000 0000
  adding the missing bit = 1.101
- Create the binary result:
  $1.101 * 2^{-5} = 0.00001101$
  = 1/32 + 1/64 + 1/256 = (8 + 4 + 1)/256
  = 13/256 = .05078125
- Don't forget the sign!
  answer = -0.05078125

- BE400000h in IEEE?

ASCII

- ASCII (American Standard Code for Information Interchange) is commonly used to represent characters sent to a display
- This is what is used to display information (letters, symbols, numbers, and control characters).
- Examples:
  'A' = 41h = 65_{10}
  'a' = 61h = 97_{10}
  '!' = 21h = 33_{10}
  '1' = 31h = 49_{10}
  CR (carriage return) = 0Dh = 13_{10}