Relevant Course Information

❖ hw5 due Wednesday, hw6 due Friday

❖ Don’t change your poll answers after-the-fact!
  ▪ Graded on completion; misrepresents your understanding

❖ Lab 1a due tonight at 11:59 pm
  ▪ Submit pointer.c and lab1Asynthesis.txt
    • Make sure there are no lingering printf statements in your code!
  ▪ Make sure you submit something to Gradescope before the deadline and that the file names are correct
  ▪ Can use late days to submit up until Wed 11:59 pm

❖ Lab 1b due next Monday (10/17)
  ▪ Submit aisle_manager.c, store_client.c, and lab1Bsynthesis.txt
Lab 1b Aside: C Macros

❖ C macros basics:
   ▪ Basic syntax is of the form: `#define NAME expression`
   ▪ Allows you to use “NAME” instead of “expression” in code
     • Does naïve copy and replace before compilation – everywhere the characters “NAME” appear in the code, the characters “expression” will now appear instead
     • NOT the same as a Java constant
   ▪ Useful to help with readability/factoring in code

❖ You’ll use C macros in Lab 1b for defining bit masks
   ▪ See Lab 1b starter code and Lecture 4 slides (card operations) for examples
Reading Review

❖ Terminology:
  - normalized scientific binary notation
  - trailing zeros
  - sign, mantissa, exponent ↔ bit fields S, M, and E
  - float, double
  - biased notation (exponent), implicit leading one (mantissa)
  - rounding errors

❖ Questions from the Reading?
Review Questions

❖ Convert $11.375_{10}$ to normalized binary scientific notation

\[
\begin{align*}
&8 + 2 + 1 + 0.25 + 0.125 \\
&2^3 + 2^1 + 2^0 + 2^{-2} + 2^{-3} = \frac{1011}{2} \cdot 011 = \left[1.011011 \times 2^3\right]
\end{align*}
\]

❖ What is the value encoded by the following floating point number?

\[
\begin{align*}
0b \ 0 \ | \ 1000 \ 0000 \ | \ 110 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \\
\text{bias} &= 2^{3-1} = 2^2 - 1 = 3 - 1 = 2 \Rightarrow 127 \\
\text{exponent} &= E - \text{bias} = 2^3 - 127 = 128 - 127 = 1 \\
\text{mantissa} &= 1.M = 1.110...0_2
\end{align*}
\]

\[
(-1)^0 \times 1.11_2 \times 2^1 = 11.1_2 = \left[3.5\right]
\]
Number Representation Revisited

❖ What can we represent in one word?
  ▪ Signed and Unsigned Integers
  ▪ Characters (ASCII)
  ▪ Addresses

❖ How do we encode the following:
  ▪ Real numbers (e.g., 3.14159)
  ▪ Very large numbers (e.g., 6.02×10^{23})
  ▪ Very small numbers (e.g., 6.626×10^{-34})
  ▪ Special numbers (e.g., ∞, NaN)
Floating Point Topics

- IEEE floating-point standard
- Floating-point operations and rounding
- Floating-point in C

There are many more details that we won’t cover
- It’s a 58-page standard...
Binary Scientific Notation (Review)

- Normalized form: exactly one digit (non-zero) to left of binary point

- Computer arithmetic that supports this called floating point due to the “floating” of the binary point
  - Declare such variable in C as float (or double)
IEEE Floating Point

- IEEE 754 (established in 1985)
  - Standard to make numerically-sensitive programs portable
  - Specifies two things: representation scheme and result of floating point operations
  - Supported by all major CPUs

- Driven by numerical concerns
  - **Scientists/natural analysts** want them to be as **real** as possible
  - **Engineers** want them to be **easy to implement** and **fast**
  - Scientists mostly won out:
    - Nice standards for rounding, overflow, underflow, but...
    - Hard to make fast in hardware
    - **Float operations can be an order of magnitude slower than integer ops**
Floating Point Encoding (Review)

❖ Use normalized, base 2 scientific notation:
  ▪ Value: \( \pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}} \)
  ▪ Bit Fields: \((-1)^S \times 1.M \times 2^{(E-bias)}\)

❖ Representation Scheme: (3 separate fields within 32 bits)
  ▪ Sign bit (0 is positive, 1 is negative)
  ▪ Mantissa (a.k.a. significand) is the fractional part of the number in normalized form and encoded in bit vector \(M\)
  ▪ Exponent weights the value by a (possibly negative) power of 2 and encoded in the bit vector \(E\)
The Exponent Field (Review)

- **Use biased notation**
  - Read exponent as unsigned, but with bias of \(2^{w-1}-1 = 127\)
  - Representable exponents roughly \(\frac{1}{2}\) positive and \(\frac{1}{2}\) negative
  - \(\text{Exp} = E - \text{bias} \iff E = \text{Exp} + \text{bias}\)
    - Exponent 0 (\(\text{Exp} = 0\)) is represented as \(E = 0b\ 0111\ 1111 = 2^7 - 1\)

- **Why biased?**
  - Now it’s a sign-and-magnitude representation!
  - Makes floating point arithmetic easier (somewhat compatible with two’s complement hardware)
The Mantissa (Fraction) Field (Review)

- **Note the implicit leading 1 in front of the M bit vector**
  - Example: 0b 0011 1111 1100 0000 0000 0000 0000 0000 is read as $1.1_2 = 1.5_{10}$, not $0.1_2 = 0.5_{10}$
  - Gives us an extra bit of precision

- **Mantissa “limits”**
  - Low values near $M = 0b0...0$ are close to $2^{E_{\text{Exp}}}$
  - High values near $M = 0b1...1$ are close to $2^{E_{\text{Exp}}+1}$

\[ (-1)^S \times (1 \cdot M) \times 2^{(E_{\text{Exp}} - \text{bias})} \]
Normalized Floating Point Conversions

- **FP → Decimal**
  1. Append the bits of $M$ to implicit leading 1 to form the mantissa.
  2. Multiply the mantissa by $2^{E - \text{bias}}$.
  3. Multiply the sign $(-1)^S$.
  4. Multiply out the exponent by shifting the binary point.
  5. Convert from binary to decimal.

- **Decimal → FP**
  1. Convert decimal to binary.
  2. Convert binary to normalized scientific notation.
  3. Encode sign as $S$ (0/1).
  4. Add the bias to exponent and encode $E$ as unsigned.
  5. The first bits after the leading 1 that fit are encoded into $M$. 
Practice Question

❖ Convert the decimal number $-7.375 = -1.11011 \times 2^2$ into floating point representation.

\[ S = 1, \ E = 2 + 127 = 129 = \text{0b}10000001, \ M = \text{0b}110110 \ldots 0 \]
\[ \text{0b}1100 0000 1110 1100 0 \ldots 0 = 0x\text{C0EC 0000} \]
Precision and Accuracy

❖ **Accuracy** is a measure of the difference between the *actual value of a number* and its computer representation

❖ **Precision** is a count of the number of bits in a computer word used to represent a value
  - Capacity for accuracy

❖ *High precision permits high accuracy but doesn’t guarantee it*
  - **Example:** `float pi = 3.14;` will be represented using all 24 bits of the mantissa (highly precise), but is only an approximation (not accurate)
Need Greater Precision?

❖ Double Precision (vs. Single Precision) in 64 bits

- C variable declared as `double`
- Exponent bias is now $2^{10} - 1 = 1023$, $bias = 2^{10} - 1$
- **Advantages:** greater precision (larger mantissa), greater range (larger exponent)
- **Disadvantages:** more bits used, slower to manipulate
Current Limitations

- Largest magnitude we can represent?
- Smallest magnitude we can represent?
  - Limited *range* due to width of *E* field
- What happens if we try to represent $2^0 + 2^{-30}$?
  - Rounding due to limited *precision*: stores $2^0$
- There is a need for *special cases*
  - How do we represent the value zero? $0 \neq \pm 1.M \times 2^{E-\text{bias}}$
  - What about $\infty$ and NaN? ???
Summary

❖ Floating point approximates real numbers:

- Handles large numbers, small numbers, special numbers
- Exponent in biased notation \((\text{bias} = 2^{w-1} - 1)\)
  - Size of exponent field determines our representable range
  - Outside of representable exponents is overflow and underflow
- Mantissa approximates fractional portion of binary point
  - Size of mantissa field determines our representable precision
  - Implicit leading 1 (normalized) except in special cases
  - Exceeding length causes rounding
Preview Question

- Find the sum of the following binary numbers in normalized scientific binary notation:

\[
1.01_2 \times 2^0 + 1.11_2 \times 2^2
\]

- 1. Match exponents
- 2. Sum mantissas
- 3. Normalize

\[
\begin{align*}
1.00101 \times 2^2 + 1.11_2 \times 2^2 &= 10.0001 \times 2^2 \\
\end{align*}
\]

The result is:

\[
1.00001 \times 2^3
\]