Floating Point
CSE 351 Winter 2024

Instructor:
Justin Hsia

Teaching Assistants:
Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

http://www.smbc-comics.com/?id=2999
Relevant Course Information

- HW4 due tonight, HW5 due Friday, HW6 due Monday

- Lesson questions are graded on completion
  - Don’t change your answer afterward; misrepresents your understanding

- Lab 1a final late submissions due tonight at 11:59 pm
  - Submit pointer.c and lab1Asynthesis.txt
  - Make sure there are no lingering printf statements in your code!

- Lab 1b due Monday (1/22)
  - 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 Lesson 4 (card operations) for examples
Floating Point
Lesson Summary (1/2)

❖ Floating point approximates real numbers (large, small, & special):

- Normalized case: $\pm 1 \times \text{Mantissa} \times 2^{\text{Exponent}} = (-1)^S \times 1.M \times 2^{(E-bias)}$

- Mantissa approximates fractional portion
  - Size of mantissa field determines our representable precision
  - Exceeding mantissa length causes rounding

- 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

<table>
<thead>
<tr>
<th>E</th>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0...0</td>
<td>anything</td>
<td>± denorm num (including 0)</td>
</tr>
<tr>
<td>anything else</td>
<td>anything</td>
<td>± norm num</td>
</tr>
<tr>
<td>0b1...1</td>
<td>0</td>
<td>± $\infty$</td>
</tr>
<tr>
<td>0b1...1</td>
<td>non-zero</td>
<td>NaN</td>
</tr>
</tbody>
</table>

- double (64 bits: $[S (1)| E (11)| M (52)]$) available if more precision needed
Lesson Summary (2/2)

❖ Limitations of FP affect programmers all the time (!)

▪ Overflow, underflow, rounding
  • Rounding is a HUGE issue due to limited mantissa bits and gaps that are scaled by the value of the exponent

▪ Floating point arithmetic is NOT associative or distributive
  • $\infty$ and NaN are valid operands, but can produce unintuitive results

▪ Do NOT use equality ($==$) with floating point numbers

▪ Converting between integral and floating point data types does change the bits
  • e.g., `int i = 2; // stored as 0x00000002, float f = i; // stored as 0x40000000`
Lesson Q&A

❖ Learning Objectives:
  ▪ Describe how the bits in floating point are organized and how they represent real numbers (and special cases).
  ▪ Describe the distribution of representable values in floating point.
  ▪ Explain the limitations of floating point and write C code that accounts for them.

❖ What lingering questions do you have from the lesson?
  ▪ Chat with your neighbors about the lesson for a few minutes to come up with questions
Floating Point – Practice
Polling Questions (1/2)

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

\[
\begin{array}{c}
0b \ 0 | \ 1000 \ 0000 | \ 110 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \\
\end{array}
\]

- bias = \(2^{8-1}-1\) = \(2^7-1=127\)
- exponent = \(E - \text{bias}\) = \(2^7 - 127 = 128 - 127 = 1\)
- mantissa = 1.M = 1.110...2

\((-1)^0 \times 1.11 \times 2^1 = 11.1_2 = +3.5\)

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

\[
\begin{array}{c}
S = 1, \ E = 2+127 = 129 = \text{Ob 100 0000}, \ M = \text{Ob 1101110...0} \\
0b \ 1100 \ 0000 \ 1110 \ 1100 \ 0000 \ 0000 = \text{0x COEC 0000} \\
\end{array}
\]
Polling Questions (2/2)

❖ What is the value of the following floats?
   - \(0x00000000\) \(\Rightarrow S=0, E=0, M=0 \Rightarrow +0\)
   - \(0xFF800000\) \(\Rightarrow S=1, E=111, M=0 \Rightarrow -\infty\)

❖ For the following code, what is the smallest value of \(n\) that will encounter a limit of representation?

```c
float f = 1.0; // 2^0
for (int i = 0; i < n; ++i)
    f *= 1024; // 1024 = 2^10
printf("f = %f\n", f);
```

\(\begin{array}{c|c}
  n & f \\
  \hline
  1 & 2^0 \\
  2 & 2^2 \\
  3 & 2^3 \\
  4 & 2^4 \\
  \vdots & \vdots \\
  \end{array}\)

\(E_{\text{max}} = 0x\text{FE}, \quad E_{\text{max}} = 2^{127} - 127 = 127\)

\(\text{for } n=133, \text{ we hit } 2^{130}, \text{ which causes overflow}\)

\(\text{always } M = 0\)
Homework Setup

- Let `float f = 1073741824; // 2^30;`
- What’s the smallest power of 2 for `g` such that `f + g != f`?
Floating Point – Context
Floating Point Issues in Real Life

❖ **1991:** Patriot missile targeting error
  ▪ Time in system stored in integer (tenths of a second since boot)
  ▪ Converted to seconds by multiplying by 0.1 = 0.0 0011₂ leading to erroneous time (error grows the longer system has been on)

❖ **1996:** V88 Ariane 501 rocket exploded 37 seconds after launch
  ▪ Reused code from Ariane 4 inertial reference platform
  ▪ Overflow when converting a 64-bit floating point number to a 16-bit integer (not protected by extra lines of code)

❖ **Other related bugs:**
  ▪ 1982: Vancouver Stock Exchange 50% error in less than 2 years due to truncation
  ▪ 1994: Intel Pentium FDIV (floating point division) hardware bug costs company $475 million in recall
More on Floating Point History

❖ Early days
  ▪ First design with floating-point arithmetic in 1914 by Leonardo Torres y Quevedo
  ▪ Implementations started in 1940 by Konrad Zuse, but with differing field lengths (usually not summing to 32 bits) and different subsets of the special cases

❖ IEEE 754 standard created in 1985
  ▪ Primary architect was William Kahan, who won a Turing Award for this work
  ▪ Standardized bit encoding, well-defined behavior for all arithmetic operations
Floating Point in the “Wild”

❖ 3 formats from IEEE 754 standard widely used in computer hardware and languages
  ▪ In C, called float, double, long double

❖ Common applications:
  ▪ 3D graphics: textures, rendering, rotation, translation
  ▪ “Big Data”: scientific computing at scale, machine learning

❖ Non-standard formats in domain-specific areas:
  ▪ **Bfloat16**: training ML models; range more valuable than precision
  ▪ **TensorFloat-32**: Nvidia-specific hardware for Tensor Core GPUs

<table>
<thead>
<tr>
<th>Type</th>
<th>S bits</th>
<th>E bits</th>
<th>M bits</th>
<th>Total bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Half-precision</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td>Bfloat16</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>TensorFloat-32</td>
<td>1</td>
<td>8</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>Single-precision</td>
<td>1</td>
<td>8</td>
<td>23</td>
<td>32</td>
</tr>
</tbody>
</table>
Discussion Question

❖ Discuss the following question(s) in groups of 3-4 students
  ▪ I will call on a few groups afterwards so please be prepared to share out
  ▪ Be respectful of others’ opinions and experiences

❖ How do you feel about floating point?
  ▪ Do you feel like the limitations are acceptable?
  ▪ Does this affect the way you’ll think about non-integer arithmetic in the future?
  ▪ Are there any changes or different encoding schemes that you think would be an improvement?
Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the lab (if applicable)

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support