Representing Data

15-213: Introduction to Computer Systems
Recitation 3: Monday, Sept. 9th, 2013
Marjorie Carlson
Section A
Welcome to 15-213 (Belatedly!)

- Yay 15-213!

- My advice on doing well in this course:
  - Labs: start early.
  - Exams: you should already be doing practice exam questions.
    - Previous exam questions and answers are all online.
    - Questions don’t change much from semester to semester.
    - If you do the exam questions related to each week’s topic as you go, you’ll know all the material by exam time.
  - The textbook is actually really useful. (!)
  - General advice: this course is a good place to get more comfortable with UNIX and C, if you aren’t already.
Welcome to Recitation

- Recitation is a place for interaction
  - If you have questions, please ask.
  - If you want to go over an example not planned for recitation, let me know.

- Each week we’ll cover:
  - A quick recap of topics from class, especially ones we have found students struggled with in the past.
  - Tips for labs.
  - Sample problems to reinforce the main ideas and prepare for exams.
Agenda

- How Do I Data Lab?
- Integers
  - Biasing division
- Floats
  - Binary fractions
  - IEEE standard
  - Example problem
How Do I Data Lab?

(due Thursday at 11:59 pm)

- **Step 1: Download lab files**
  - All lab files are on autolab
  - Remember to also read the lab handout (“view writeup” link)

- **Step 2: Work on the right machines**
  - Remember to do all your lab work on Shark machines
  - **This includes untarring the handout.** Otherwise, you may lose some permissions bits
  - If you get a permission denied error, try “chmod +x filename”
  - Do your work in bits.c
How Do I Data Lab?

- **Step 3: Test test test!**
  We have given you FOUR WAYS to test your code before submitting!
  - `.btest` lets you debug (printf, test single inputs).
    Type `make` before using it.
  - `.dlc bits.c` enforces the coding rules (number of operations).
  - `.bddcheck/check.pl` tests *definitely* for correctness.
  - `.driver.pl` uses both DLC and the BDD checker – this is what Autolab uses.

- **Code that passes btest will not necessarily pass autolab!**
How Do I Data Lab?

- **Step 4: Submit to Autolab**
  - Unlimited submissions, but please don’t use autolab in place of driver.pl
  - Must submit via web form
  - To package/download files to your computer, use `tar -cvzf out.tar.gz in1 in2 ...` (if relevant) and your favorite file transfer protocol
How Do I Data Lab? – Tips!

- Write C like it’s 1989 (for DLC – only used in data lab)
  - Declare variable at top of function
  - Make sure closing brace (”}”) is in 1st column

- Be careful of operator precedence
  - Do you know what order ~a+1+b*c<<3*2 will execute in?
  - Neither do I. Use parentheses: (~a)+1+(b*(c<<3)*2)

- Take advantage of special values like 0, -1, and T_min

- Operations with undefined behavior in C may have defined behavior on our architecture. (Examples: addition overflow, bit-shifting by 32.) It’s OK to use them.

- Reducing operations once you’re under the threshold won’t get you extra points (just more glory).
Where Can I Get Help?

- The assignment writeup
- The assignment writeup!
- The assignment writeup!!!!!!!!
- Lecture notes and the textbook
- Staff email list: [15-213-staff@cs.cmu.edu](mailto:15-213-staff@cs.cmu.edu)
- Office hours: Sun-Thu, 5:30-8:30 pm, in Wean 5027
- Peer tutoring: Tue 8:30-11, Mudge Reading Room
Agenda

- How Do I Data Lab?
- **Integers**
  - Biasing division
- **Floats**
  - Binary fractions
  - IEEE standard
  - Example problem
Integers – Biasing

- You can multiply and divide by powers of 2 with bitshifts
- As you’ll see when we learn assembly, your computer does this a lot!
  - Multiply:
    - Left shift by k to multiply by $2^k$
    - Let’s try this with binary 00010
  - Divide:
    - Right shift by k to divide by $2^k$... sort of
    - Let’s try this with binary 01111
    - How about binary 10001
    - Uh-oh!
    - Shifting rounds down, but we want to round toward zero.
    - Solution: biasing when the number is negative
Integers – Biasing

Remember biasing flips rounding direction; only use when dividend is negative

If this contains a 1...

Biasing adds 1 to final result
Agenda

- How Do I Data Lab?
- Integers
  - Biasing division
- Floats
  - Binary fractions
  - IEEE standard
  - Example problem
Floating Point – Fractions in Binary

- Representation
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number:
    \[ \sum_{k=-j}^{i} b_k \times 2^k \]
Floating Point – Fractions in Binary

- Convert binary to decimal:
  - 1.1
  - 0.0011
  - 1010.00101

- Convert decimal to binary:
  - 3 3/4
  - 2 3/32
  - 5.875
How Can We Represent Numbers Efficiently?

What do we do if we want to convey

-592349235823740180.3

in 10 digits?
How Can We Represent Numbers Efficiently?

What do we do if we want to convey

\[-592349235823740180.3\]

in 10 digits?

Hint:

\[\text{---*---}\]
How Can We Represent Numbers Efficiently?

What do we do if we want to convey

\[-592349235823740180.3\]

in 10 digits?

\[-5.92349 \times 10^{17}\]
Floating Point – Scientific Notation

- So, how can we put binary numbers into scientific notation?

\[ 101.111 \]

\[ 1.01111 \times 2^2 \]

- Numerical form: \((-1)^S M 2^E\)
Floating Point – IEEE Standard

- Floating points are basically a way to encode binary scientific notation.

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

- But exp ≠ E and frac ≠ M, because IEEE format optimizes to increase the range of numbers that can be represented.
  - If numbers are always in the format 1.xxxx (we’ll revisit this!), encoding the 1 is unnecessary. So frac is simply M without the leading 1. \( M = 1 + \text{frac} \)
  - exp is unsigned and can represent the numbers 0 to 255. We’d rather have it represent -127(ish) to 128(ish), so we subtract a bias of 127 \( (2^{k-1}-1) \) get from E to get exp. \( E = \text{exp} - \text{bias} \)
Floating Point – IEEE Standard

101.111

1.01111 * 2^2

- S = +
- E = 2
- M = 1.01111

so s = 0
so exp =
so frac =

Remember!
M = 1 + frac
E = exp - bias
Bias = 127
Floating Point – IEEE Standard

\[ 101.111 \]

\[ 1.01111 \times 2^2 \]

- **Sign (S)**: +
- **Mantissa (M)**: 1.01111
- **Exponent (E)**: 2

\[ \text{so } s = 0 \]
\[ \text{so exp} = 129 \ (2 + \text{bias}) \]
\[ \text{M} = 1.01111 \]
\[ \text{so frac} = \]

**Remember!**
- \( M = 1 + \text{frac} \)
- \( E = \text{exp} - \text{bias} \)
- \( \text{Bias} = 127 \)
Floating Point – IEEE Standard

\[ 101.111 \]

\[ 1.01111 \times 2^2 \]

- \( S = + \) so \( s = 0 \)
- \( E = 2 \) so \( \text{exp} = 129 \ (2 + \text{bias}) \)
- \( M = 1.01111 \) so \( \text{frac} = 01111 \)

Remember!
- \( M = 1 + \text{frac} \)
- \( E = \text{exp} - \text{bias} \)
- \( \text{Bias} = 127 \)
Floating Point – Example

Consider the following 5-bit floating point representation based on the IEEE floating point format. This format does not have a sign bit – it can only represent nonnegative numbers.

- There are $k=3$ exponent bits.
- There are $n=2$ fraction bits.

What’s the bias?

- What does 100 10 represent?
- What does 001 01 represent?
- How would you represent 6?
- How would you represent $\frac{1}{4}$?
Floating Point – Denormalized Range

- If that was all there was to it, the smallest number representable would be $2^{-\text{bias}}$, which is not that small. And it would be represented by $000\ 00$. Hmm...
- IEEE uses a trick to give us numbers closer to 0: drop the implied leading 1.

<table>
<thead>
<tr>
<th>Normalized</th>
<th>Denormalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp $\neq$ 0</td>
<td>exp $= 0$</td>
</tr>
<tr>
<td>implied leading 1</td>
<td>no implied leading 1</td>
</tr>
<tr>
<td>$E = \text{exp} - \text{bias}$</td>
<td>$E = 1 - \text{bias}$</td>
</tr>
<tr>
<td>denser near origin</td>
<td>evenly spaced</td>
</tr>
<tr>
<td>represents most numbers</td>
<td>represents very small numbers</td>
</tr>
</tbody>
</table>
Floating Point – Special Cases

- Well, denormalizing got us our 0. Now how about infinity?

- The largest exponent is coopted to encode special cases:
  - $\exp = \text{all } 1\text{s}$$\frac{\text{frac}}{} = \text{all } 0\text{s}$$\text{represents infinity (} + \text{ or } -\text{)}$
  - $\exp = \text{all } 1\text{s}$$\frac{\text{frac}}{} \text{isn’t all } 0\text{s}$$\text{represents NaN}$
Floating Point – Special Case Examples

- Back to our mini-floats:
  - There are $k=3$ exponent bits.
  - There are $n=2$ fraction bits.
  - Bias = 3

- What does 000 10 represent?
- What’s the smallest representable nonzero value?
- What’s the largest representable finite number?
- What’s the smallest normalized number?
- What’s the largest denormalized number?
Two More Tips and You Can Convert Anything!

- Decimal → float is a little trickier because you have to figure out whether it has to be encoded as normalized or denormalized.
  - Strategy 1: compare your number to the smallest normalized number before converting it.
  - Strategy 2: try to encode it as normalized; if your exponent doesn’t fit in exp, change exp to 0 and shift your decimal point accordingly.

- You need to know how to round!
Floating Point – Rounding

- Floats round to even
  - Why? Avoid statistical bias of always rounding up or down.
  - How? Like this:

    1.0100\textsubscript{2} truncate 1.01\textsubscript{2}
    1.0101\textsubscript{2} below half; round down 1.01\textsubscript{2}
    1.0110\textsubscript{2} interesting case; round to even 1.10\textsubscript{2}
    1.0111\textsubscript{2} above half; round up 1.10\textsubscript{2}
    1.1000\textsubscript{2} truncate 1.10\textsubscript{2}
    1.1001\textsubscript{2} below half; round down 1.10\textsubscript{2}
    1.1010\textsubscript{2} Interesting case; round to even 1.10\textsubscript{2}
    1.1011\textsubscript{2} above half; round up 1.11\textsubscript{2}
    1.1100\textsubscript{2} truncate 1.11\textsubscript{2}
Floating Point – Rounding Examples

Back to our mini-floats:

- There are $k=3$ exponent bits.
- There are $n=2$ fraction bits.
- Bias = 3

<table>
<thead>
<tr>
<th>Value</th>
<th>Floating Point Bits</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Floating Point – Rounding Examples

- Back to our mini-floats:
  - There are k=3 exponent bits.
  - There are n=2 fraction bits.
  - Bias = 3

<table>
<thead>
<tr>
<th>Value</th>
<th>Floating Point Bits</th>
<th>Rounded Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/32</td>
<td>001 00</td>
<td>1/4</td>
</tr>
<tr>
<td>8</td>
<td>110 00</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>110 00</td>
<td>8</td>
</tr>
<tr>
<td>19</td>
<td>111 00</td>
<td>+ inf</td>
</tr>
</tbody>
</table>
Questions?