Representing Data

15-213: Introduction to Computer Systems
Recitation 3: Monday, Sept. 9th, 2013
Ian Hartwig
Section E
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.
  - We don’t spend a lot of class time on using UNIX/C. Is this needed?

- We’ll cover:
  - A quick recap of topics from class, especially ones we have found students struggled with in the past
  - Example problems to reinforce those topics and prepare for exams
  - Tips for labs
News

- Tutoring available in Mudge Reading Room, Tues. 8:30-11pm.
- Office hours in WeH 5207, Sun.-Thur. 5:30-8:30pm
- Data lab due this Thursday at 11:59pm
Agenda

- How do I Data Lab?
- Integers
  - Biasing division
  - Endianness
- Floating point
  - Binary fractions
  - IEEE standard
  - Example problem
How do I Data Lab?

- **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 Andrew or Shark machines
    - Some later labs will restrict you to just the shark machines (bomb lab, for example)
  - This includes untaring the handout. Otherwise, you may lose some permissions bits
  - If you get a permission denied error, try “chmod +x filename”
How do I Data Lab?

- **Step 3: Edit and test**
  - Bits.c is the file you’re looking for
  - Remember you have 3 ways to test your solutions. See the writeup for details.
  - driver.pl runs the same tests as autolab

- **Step 4: Submit**
  - 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 …” and your favorite file transfer protocol
How do I Data Lab?

Tips

- Write C like it’s 1989
  - Declare variable at top of function
  - Make sure closing brace ("\}") is in 1st column
  - We won’t be using the dlc compiler for later labs
- Be careful of operator precedence
  - Do you know what order \(\sim a + 1 + b \times c << 3 \times 2\) will execute in?
  - Neither do I. Use parentheses: \((\sim a) + 1 + (b \times (c << 3) \times 2)\)
- Take advantage of special operators and values like !, 0, and \(T_{\text{min}}\)
- Reducing ops once you’re under the threshold won’t get you extra points.
- Undefined behavior
  - Like shifting by <32. See Anita’s rant.
Anita’s Rant

**From the Intel x86 Reference:**

“These instructions shift the bits in the first operand (destination operand) to the left or right by the number of bits specified in the second operand (count operand). *Bits shifted beyond the destination operand boundary are first shifted into the CF flag, then discarded.* At the end of the shift operation, the CF flag contains the last bit shifted out of the destination operand.

The destination operand can be a register or a memory location. The count operand can be an immediate value or register CL. *The count is masked to five bits, which limits the count range to 0 to 31.* A special opcode encoding is provided for a count of 1.”
Integers - Biasing

- Can multiply/divide powers of 2 with shift
  - Multiply:
    - Left shift by $k$ to multiply by $2^k$
  - Divide:
    - Right shift by $k$ to divide by $2^k$
    - ... for positive numbers
    - Shifting rounds towards $-\infty$, but we want to round to 0
    - Solution: biasing when 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
Integers – Endianness

- Endianness describes which bit is most significant in a binary number
- You won’t need to work with this until bomb lab
- Big endian:
  - First byte (lowest address) is the most significant
  - This is how we typically talk about binary numbers
- Little endian:
  - First byte (lowest address) is the least significant
  - Intel x86 (shark/andrew linux machines) implement this
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 – IEEE Standard

- Single precision: 32 bits
  
<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>

- Double precision: 64 bits
  
<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11-bits</td>
<td>52-bits</td>
</tr>
</tbody>
</table>

- Extended precision: 80 bits (Intel only)
  
<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15-bits</td>
<td>63 or 64-bits</td>
</tr>
</tbody>
</table>
Floating Point – IEEE Standard

What does this mean?
- We can think of floating point as binary scientific notation
  - IEEE format includes a few optimizations to increase range for our given number of bits
  - The number represented is essentially \( (\text{sign} \times \text{frac} \times 2^{\text{exp}}) \)
    - There are a few steps I left out there

Example:
- Assume our floating point format has no sign bit, \( k = 3 \) exponent bits, and \( n=2 \) fraction bits
- What does \texttt{0b10010} represent?
Floating Point – IEEE Standard

■ What does this mean?
  ▪ We can think of floating point as binary scientific notation
    ▪ IEEE format includes a few optimizations to increase range for our given number of bits
  ▪ The number represented is *essentially* \((\text{sign} \times \frac{\text{frac}}{2^\text{exp}})\)
    ▪ There are a few steps I left out there

■ Example:
  ▪ Assume our floating point format has no sign bit, \(k = 3\) exponent bits, and \(n=2\) fraction bits
  ▪ What does 0b10010 represent? 3
# Floating Point – IEEE Standard

- **Bias**
  - exp is unsigned; needs a bias to represent negative numbers
  - \( \text{Bias} = 2^{k-1} - 1 \), where \( k \) is the number of exponent bits
  - Can also be thought of as bit pattern \( 0b011...111 \)

- **Normalized**

<table>
<thead>
<tr>
<th></th>
<th><strong>Denormalized</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied leading 1</td>
<td>exp = 0</td>
</tr>
<tr>
<td>( E = \text{exp} - \text{Bias} )</td>
<td></td>
</tr>
<tr>
<td>Denser near origin</td>
<td>Represents small numbers</td>
</tr>
</tbody>
</table>

- When converting frac/int => float, assume normalized until proven otherwise
# Floating Point – IEEE Standard

- **Bias**
  - exp is unsigned; needs a bias to represent negative numbers
  - Bias = $2^{k-1} - 1$, where k is the number of exponent bits
  - Can also be thought of as bit pattern 0b011...111

- **Normalized**

<table>
<thead>
<tr>
<th>0 &lt; exp &lt; $(2^k-1)$</th>
<th>Denormalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implied leading 1</td>
<td>exp = 0</td>
</tr>
<tr>
<td>E = exp - Bias</td>
<td>Leading 0</td>
</tr>
<tr>
<td>Denser near origin</td>
<td>E = 1 - Bias. Why?</td>
</tr>
<tr>
<td><strong>Represents large numbers</strong></td>
<td><strong>Evenly spaced</strong></td>
</tr>
<tr>
<td>Represents small numbers</td>
<td></td>
</tr>
</tbody>
</table>

- When converting frac/int => float, assume normalized until proven otherwise
Floating Point – IEEE Standard

- Special Cases (exp = 2^{k-1})
  - Infinity
    - Result of an overflow during calculation or division by 0
    - exp = 2^{k-1}, frac = 0
  - Not a Number (NaN)
    - Result of illegal operation (sqrt(-1), inf – inf, inf * 0)
    - exp = 2^{k-1}, frac != 0
  - Keep in mind these special cases are not the same
## Floating Point – IEEE Standard

- **Round to even**
  - **Why?** Avoid statistical bias of rounding up or down on half.
  - **How?** Like this:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Fraction</th>
<th>Action</th>
<th>Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0100</td>
<td></td>
<td>truncate</td>
<td>1.01</td>
</tr>
<tr>
<td>1.0101</td>
<td></td>
<td>below half; round down</td>
<td>1.01</td>
</tr>
<tr>
<td>1.0110</td>
<td></td>
<td>interesting case; round to even</td>
<td>1.10</td>
</tr>
<tr>
<td>1.0111</td>
<td></td>
<td>above half; round up</td>
<td>1.10</td>
</tr>
<tr>
<td>1.1000</td>
<td></td>
<td>truncate</td>
<td>1.10</td>
</tr>
<tr>
<td>1.1001</td>
<td></td>
<td>below half; round down</td>
<td>1.10</td>
</tr>
<tr>
<td>1.1010</td>
<td></td>
<td>Interesting case; round to even</td>
<td>1.10</td>
</tr>
<tr>
<td>1.1011</td>
<td></td>
<td>above half; round up</td>
<td>1.11</td>
</tr>
<tr>
<td>1.1100</td>
<td></td>
<td>truncate</td>
<td>1.11</td>
</tr>
</tbody>
</table>
Rounding

Guard bit: LSB of result
Round bit: 1\textsuperscript{st} bit removed
Sticky bit: OR of remaining bits

- Round up conditions
  - Round = 1, Sticky = 1 $\rightarrow$ > 0.5
  - Guard = 1, Round = 1, Sticky = 0 $\rightarrow$ Round to even

<table>
<thead>
<tr>
<th>Value</th>
<th>Fraction</th>
<th>GRS</th>
<th>Incr?</th>
<th>Rounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.0000000</td>
<td>000</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>15</td>
<td>1.1010000</td>
<td>100</td>
<td>N</td>
<td>1.101</td>
</tr>
<tr>
<td>17</td>
<td>1.0001000</td>
<td>010</td>
<td>N</td>
<td>1.000</td>
</tr>
<tr>
<td>19</td>
<td>1.0011000</td>
<td>110</td>
<td>Y</td>
<td>1.010</td>
</tr>
<tr>
<td>138</td>
<td>1.0001010</td>
<td>011</td>
<td>Y</td>
<td>1.001</td>
</tr>
<tr>
<td>63</td>
<td>1.1111100</td>
<td>111</td>
<td>Y</td>
<td>10.000</td>
</tr>
</tbody>
</table>
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 is the...
  - Bias?
  - Largest denormalized number?
  - Smallest normalized number?
  - Largest finite number it can represent?
  - Smallest non-zero value it can represent?
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 is the...

- Bias? $011_2 = 3$
- Largest denormalized number? $000\ 11_2 = 0.0011_2 = 3/16$
- Smallest normalized number? $001\ 00_2 = 0.0100_2 = 1/4$
- Largest finite number it can represent? $110\ 11_2 = 1110.0_2 = 14$
- Smallest non-zero value it can represent? $000\ 01_2 = 0.0001_2 = 1/16$
**Floating Point – Example**

For the same problem, complete the following table:

<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></td>
<td>000 10</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Floating Point – Example

- For the same problem, complete the following table:

<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>1/8</td>
<td>000 10</td>
<td>inf</td>
</tr>
<tr>
<td>19</td>
<td>111 00</td>
<td>inf</td>
</tr>
</tbody>
</table>
Floating Point Recap

- Floating point = \((-1)^s \times M \times 2^E\)
- MSB is sign bit \(s\)
- Bias = \(2^{(k-1)} - 1\) (\(k\) is num of \(\text{exp}\) bits)
- Normalized (larger numbers, denser towards 0)
  - \(\text{exp} \neq 000...0\) and \(\text{exp} \neq 111...1\)
  - \(M = 1.\text{frac}\)
  - \(E = \text{exp} - \text{Bias}\)
- Denormalized (smaller numbers, evenly spread)
  - \(\text{exp} = 000....0\)
  - \(M = 0.\text{frac}\)
  - \(E = -\text{Bias} + 1\)
Floating Point Recap

- **Special Cases**
  - +/- Infinity: \(\text{exp} = 111\ldots1\) and \(\text{frac} = 000\ldots0\)
  - +/- NaN: \(\text{exp} = 111\ldots1\) and \(\text{frac} \neq 000\ldots0\)
  - +0: \(s = 0, \text{exp} = 000\ldots0\) and \(\text{frac} = 000\ldots0\)
  - -0: \(s = 1, \text{exp} = 000\ldots0\) and \(\text{frac} = 000\ldots0\)

- **Round towards even when half way** (lsb of \(\text{frac} = 0\))
Questions?