Anita’s Super Awesome Recitation Slides
15/18-213: Introduction to Computer Systems Datalab and Floating Point, 28 Jan 2013
Anita Zhang, Section M
FAQ and Administrative Stuff

- Staff: 15-213-staff@cs.cmu.edu
- Data lab due this Thursday, 31 Jan 2013
- FAQ on the main site
- Questions?
- Progress?
- Autolab?
- Shark? > ssh shark.ics.cs.cmu.edu
Because everyone needs a guide.

- Literature
- Bits and Bytes
- IEEE Floating Point
  - Review
  - Examples from past midterms
- Datalab Hints
- Special Notice
- More Question Time
Books I Like


There are only 10 kinds of people.
Those who understand binary and those who don’t.
RANDOM INT STUFF

- Multiplication/ division by $2^k$
  - Multiply => left shift by $k$
  - Division => right shift by $k$
RANDOM INT STUFF

...I lied. Kind of.
**Random int Stuff**

- Division of a negative number by $2^k$
  - Needs a bias
  - Looks like this: $(x + (1 << k) - 1) >> k$
RANDOM int STUFF

- Endianness is real
- But you won’t see it in datalab

<table>
<thead>
<tr>
<th>Endian</th>
<th>First byte (lowest address)</th>
<th>Middle bytes</th>
<th>Last byte (highest address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>big</td>
<td>Most significant</td>
<td>...</td>
<td>leastsignificant</td>
</tr>
<tr>
<td>little</td>
<td>Least significant</td>
<td>...</td>
<td>most significant</td>
</tr>
</tbody>
</table>

- Random example: 0x59645322
  - Big: 59 64 53 22
  - Little: 22 53 64 59
(Quick and dirty) Floating Point

For single precision (32 bit) floating point:
- Fraction (frac): 23 bits
- Exponent (exp): 8 bits
- Sign (s): 1 bit
- Bias = 127

In general:
- Value = \((-1)^s \times M \times 2^E\)
- Bias = \(2^{k-1} - 1\)
**Quick and Dirty Floating Point**

<table>
<thead>
<tr>
<th>Normalized</th>
<th>Denormalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>exp $\neq 00...0$</td>
<td>exp $= 00..0$</td>
</tr>
<tr>
<td>exp $\neq 11...1$</td>
<td>E $= 1 - \text{bias}$</td>
</tr>
<tr>
<td>$E = \exp - \text{bias}$</td>
<td>$M = 0.xx\ldots\ldots$</td>
</tr>
<tr>
<td>$M = 1.xx\ldots\ldots$</td>
<td>$\text{xxxxx is the frac}$</td>
</tr>
<tr>
<td></td>
<td>$\text{xxxxx is the frac}$</td>
</tr>
<tr>
<td></td>
<td>Leading 0</td>
</tr>
<tr>
<td></td>
<td>frac $= 0$ means $\pm 0$</td>
</tr>
</tbody>
</table>

- Implied leading 1
SPECIAL CASES

Infinity

- exp = 11....1
- frac = 00...0
  - Division by 0, ± ∞

Not a Number

- exp = 11....1
- frac =/= 00...0
  - sqrt(-1), ∞ - ∞, ∞ x 0
SPECIAL CASES

- BTW, infinity and NaN are not the same.
  - Be aware of this for float_abs()
LEGIT FLOATING POINT RULES

- **Rounding**
  - Rounds to even
    - To avoid statistical bias
    - $1.1011 \Rightarrow 1.11$ (>1/2 up)
    - $1.1010 \Rightarrow 1.10$ (1/2 down)
    - $1.0101 \Rightarrow 1.01$ (>1/2 down)
    - $1.0110 \Rightarrow 1.10$ (1/2 up)

- **Addition and Multiplication...**
  - Are lies
    - Associativity/ distributivity may not hold
    - $3.14 + (1e20 - 1e20)$ vs. $(3.14 + 1e20) - 1e20$
FLOATING POINT ON EXAMS

Let’s pretend we have a 5-bit floating point representation with no sign bit... (sadness)
- k = 3 exponent bits (bias = 3)
- n = 2 fraction bits

<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>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15/2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s pretend we have a 5-bit floating point representation with no sign bit... (sadness)

- $k = 3$ exponent bits (bias = 3)
- $n = 2$ fraction bits

<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>3</td>
<td>100 10</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>110 00</td>
<td>8</td>
</tr>
<tr>
<td>3/16</td>
<td>000 11</td>
<td>3/16</td>
</tr>
<tr>
<td>15/2</td>
<td>110 00</td>
<td>8</td>
</tr>
</tbody>
</table>
Consider two 7 bit floating point representations based on the IEEE format. Neither has a sign bit.

- **Format A**
  - $k = 3$ exponent bits (bias = 3)
  - $n = 4$ fraction bits

- **Format B**
  - $k = 4$ exponent bits (bias = 7)
  - $n = 3$ fraction bits

<table>
<thead>
<tr>
<th>Format A</th>
<th>Format B</th>
</tr>
</thead>
<tbody>
<tr>
<td>011 0000</td>
<td>0111 000</td>
</tr>
<tr>
<td>101 1110</td>
<td></td>
</tr>
<tr>
<td>010 1001</td>
<td></td>
</tr>
<tr>
<td>110 1111</td>
<td></td>
</tr>
<tr>
<td>000 0001</td>
<td></td>
</tr>
</tbody>
</table>
FLOATING POINT ON EXAMS

- Consider two 7 bit floating point representations based on the IEEE format. Neither has a sign bit.

- Format A
  - $k = 3$ exponent bits (bias = 3)
  - $n = 4$ fraction bits

- Format B
  - $k = 4$ exponent bits (bias = 7)
  - $n = 3$ fraction bits

<table>
<thead>
<tr>
<th>Format A</th>
<th>Format B</th>
</tr>
</thead>
<tbody>
<tr>
<td>011 0000</td>
<td>0111 000</td>
</tr>
<tr>
<td>101 1110</td>
<td>1001 111</td>
</tr>
<tr>
<td>010 1001</td>
<td>0110 100</td>
</tr>
<tr>
<td>110 1111</td>
<td>1011 000</td>
</tr>
<tr>
<td>000 0001</td>
<td>0001 000</td>
</tr>
</tbody>
</table>
Datalab Other Stuff

- Use the tools
  - ./driver.pl
  - ./bddcheck/check.pl
    - Exhaustive
  - ./btest
    - Not exhaustive
  - ./dlc
    - This one will hate you if you’re not writing C like it’s 1989
    - Declare all your variables at the beginning of the function
Datalab Other Stuff

- Operator Precedence
  - There are charts. Google them.

- `bang()`
  - “What’s so special about 0 and $T_{\text{min}}$ anyways?”

- `subOK()`
  - When is it possible to overflow with subtraction? Addition?

- Undefined behavior
  - Shifting by 32... omg /* insert rant here */
UNDEFINED BEHAVIOR (ADV. TOPIC)

“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.”
**Warning**

- TAs reserve the right to dock style points for code that is poorly commented, has no sense of variable names, lacks any form of coherent indentation, and of course, contains lines that are longer than 80 characters.
Questions & Credits Page

- [http://www.supersilkscreen.com](http://www.supersilkscreen.com)
- [http://www.cs.cmu.edu/~213/](http://www.cs.cmu.edu/~213/)
- Intel x86 Instruction Set Reference