Will you light my candle?

Besides getting a 1-lecture relief from listening to Matt, what was the point of Monday’s lecture?
Creating Floating Point Number

Steps

- Normalize to have leading 1
- Round to fit within fraction
- Postnormalize to deal with effects of rounding

Case Study

- Convert 8-bit unsigned numbers to tiny floating point format
- Example Numbers:
  - 128 10000000 0 1110 000
  - 138 10001010 0 1111 000
<table>
<thead>
<tr>
<th>Value</th>
<th>Binary</th>
<th>Floating-Point Value</th>
<th>Scientific Notation</th>
<th>Decimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1000 0000</td>
<td>1.000 x 2^7</td>
<td>1 x 2^7</td>
<td>128</td>
</tr>
<tr>
<td>138</td>
<td>1000 1010</td>
<td>1.001 x 2^7</td>
<td>(1+1/8) x 2^7</td>
<td>144</td>
</tr>
<tr>
<td>130</td>
<td>1000 0010</td>
<td>1.000 x 2^7</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>1000 0111</td>
<td>1.000 x 2^7</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>1000 1011</td>
<td>1.001 x 2^7</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>1001 1000</td>
<td>1.010 x 2^7</td>
<td>(1+1/4) x 2^7</td>
<td>160</td>
</tr>
</tbody>
</table>
2.84a (will look at b later)

k-bit exponent, n-bit fraction

A) V=7.0

E = 2, M=1.11
frac = 1100...0
Exp = 1000...01

bias = 2^{(e-1)}-1 = 2^{(k-1)}-1 = 0111...1

To represent E=0, need to have Exp=bias: 0111...1
To represent E=2, need to have Exp=bias+10 = 1000...1
FP Multiplication

\((-1)^{s_1} M_1 \ 2^{E_1} \ x \ (-1)^{s_2} M_2 \ 2^{E_2}\)

- **Exact Result:** \((-1)^{s} M \ 2^{E}\)
  - Sign \(s\): \(s_1 \^ s_2\)
  - Significand \(M\): \(M_1 \ * \ M_2\)
  - Exponent \(E\): \(E_1 + E_2\)

- **Fixing**
  - If \(M \geq 2\), shift \(M\) right, increment \(E\)
  - If \(E\) out of range, overflow
  - Round \(M\) to fit `frac` precision

- **Implementation**
  - Biggest chore is multiplying significands
Mathematical Properties of FP Mult

- **Compare to Commutative Ring**
  - Closed under multiplication?
  - Multiplication Commutative?
  - Multiplication is Associative?
  - 1 is multiplicative identity?
  - Multiplication distributes over addition?

- **Monotonicity**
  - $a \geq b \land c \geq 0 \Rightarrow a \cdot c \geq b \cdot c$?
Mathematical Properties of FP Mult

■ Compare to Commutative Ring
  ▪ Closed under multiplication?  Yes
    ▪ But may generate infinity or NaN
  ▪ Multiplication Commutative?  Yes
  ▪ Multiplication is Associative?  No
    ▪ Possibility of overflow, inexactness of rounding
  ▪ 1 is multiplicative identity?  Yes
  ▪ Multiplication distributes over addition?  No
    ▪ Possibility of overflow, inexactness of rounding

■ Monotonicity
  ▪ \( a \geq b \) & \( c \geq 0 \) \( \Rightarrow \) \( a \times c \geq b \times c \)?  Almost
    ▪ Except for infinities & NaNs
Floating Point Addition

\[ (-1)^{s_1} M_1 \ 2^{E_1} \ + \ (-1)^{s_2} M_2 \ 2^{E_2} \]

Assume \( E_1 > E_2 \)

**Exact Result:** \( (-1)^s \ M \ 2^E \)
- Sign \( s \), significand \( M \):
  - Result of signed align & add
- Exponent \( E \): \( E_1 \)

**Fixing**
- If \( M \geq 2 \), shift \( M \) right, increment \( E \)
- if \( M < 1 \), shift \( M \) left \( k \) positions, decrement \( E \) by \( k \)
- Overflow if \( E \) out of range
- Round \( M \) to fit frac precision
Mathematical Properties of FP Add

- Compare to those of Abelian Group
  - Closed under addition? Yes
    - But may generate infinity or NaN
  - Commutative? Yes
  - Associative? No
    - Overflow and inexactness of rounding
  - 0 is additive identity? Yes
  - Every element has additive inverse Almost
    - Except for infinities & NaNs

- Monotonicity
  - $a \geq b \Rightarrow a+c \geq b+c?$ Almost
    - Except for infinities & NaNs
Floating Point in C

- **C Guarantees Two Levels**
  - `float` single precision
  - `double` double precision

- **Conversions/Casting**
  - Casting between `int`, `float`, and `double` changes bit representation
  - `Double/float → int`
    - Truncates fractional part
    - Like rounding toward zero
    - Not defined when out of range or NaN: Generally sets to Tmin
  - `int → double`
    - Exact conversion, as long as int has $\leq 53$ bit word size
  - `int → float`
    - Will round according to rounding mode
Floating Point Puzzles

- For each of the following C expressions, either:
  - Argue that it is true for all argument values
  - Explain why not true

1. $x == (\text{int})(\text{double}) x$
2. $x == (\text{int})(\text{float}) x$
3. $f == (\text{float})(\text{double}) f$
4. $d == (\text{double})(\text{float}) d$
5. $f == -(-f)$;
6. $1.0/2 == 1/2.0$
7. $d \times d >= 0.0$
8. $(f+d)-f == d$

int x = ...;
float f = ...;
double d = ...;

Assume neither $d$ nor $f$ is NaN
Floating Point Puzzles

• For each of the following C expressions, either:
  – Argue that it is true for all argument values
  – Explain why not true

1. \( x \) == \( \text{(int)}(\text{double}) \) \( x \)  
2. \( x \) == \( \text{(int)}(\text{float}) \) \( x \)  
3. \( f \) == \( \text{(float)}(\text{double}) \) \( f \)  
4. \( d \) == \( \text{(double)}(\text{float}) \) \( d \)  
5. \( f \) == \( -(-f) \)  
6. 1.0/2 == 1/2.0  
7. \( d \times d \) >= 0.0  
8. \( (f+d)-f \) == \( d \)

Assume neither \( d \) nor \( f \) is NaN

```
int x = ...;
float f = ...;
double d = ...;
```
• 2.88 a, b-e
Moore’s “Law”

• # transistors per chip would double every year for next 10 years
  – Turns out, it’s more like every 18 month

• Storage also follows an exponential curve

• What critical improvements are sub-exponential (linear)?
Turning C into Object Code

- Code in files  `p1.c   p2.c`
- Compile with command:  `gcc -O p1.c  p2.c  -o  p`
  - Use optimizations (`-O`)
  - Put resulting binary in file `p`

\[
\begin{align*}
\text{text} & \quad \rightarrow \quad \text{C program (p1.c  p2.c)} \\
\text{text} & \quad \rightarrow \quad \text{Asm program (p1.s  p2.s)} \\
\text{binary} & \quad \rightarrow \quad \text{Object program (p1.o  p2.o)} \\
\text{binary} & \quad \rightarrow \quad \text{Executable program (p)} \\
\end{align*}
\]