Representations 2

Prof. David August
COS 217

Today
• Unsigned Multiplication
• Fixed Point
• Floating Point

Multiplication
Computing Exact Product of w-bit numbers x, y

• Need 2w bits
Unsigned: \( 0 \leq x \times y \leq (2^w - 1)^2 = 2^{2w} - 2^{w+1} + 1 \)

Two’s Complement:
min: \( x \times y \geq (-2^{w-1})(2^{w-1} - 1) = -2^{2w-2} + 2^{w-1} \)
max: \( x \times y \leq (-2^{w-1})^2 = 2^{2w-2} \)

• Maintaining Exact Results
  ○ Need unbounded representation size
  ○ Done in software by arbitrary precision arithmetic packages
  ○ Also implemented in Lisp, ML, and other languages
Unsigned Multiplication in C

- **Standard Multiplication Function**
  - Ignores high order \(w\) bits
- **Implements Modular Arithmetic**
  - \(\text{UMult}_w(u, v) = u \cdot v \mod 2^w\)

- What about unsigned integer division?

---

Unsigned Multiplication

**Binary makes it easy:**

- 0 => place 0 (0 x multiplicand)
- 1 => place a copy (1 x multiplicand)

**Key sub-parts:**

- Place a copy or not
- Shift copies appropriately
- Final addition
Representations

What can be represented in N bits?

Unsigned: \( 0 \rightarrow 2^{n-1} \)
Signed: \(-2^{n-1} \rightarrow 2^{n-1} - 1\)

What about:

Very large numbers? 9,349,787,762,244,859,087,678
Very small numbers? 0.000000000000000000004691
Rationals? 2/3
Irrationals? \(\sqrt{2}\)
Transcendentals? \(e, \pi\)

Interpretations

What should we do? Another method?

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Method 1</th>
<th>Method 2</th>
<th>Method 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>1</td>
<td>0.1</td>
</tr>
<tr>
<td>010</td>
<td>(e)</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>011</td>
<td>(\pi)</td>
<td>4</td>
<td>0.3</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>8</td>
<td>0.4</td>
</tr>
<tr>
<td>101</td>
<td>(-\pi)</td>
<td>16</td>
<td>0.5</td>
</tr>
<tr>
<td>110</td>
<td>(-e)</td>
<td>32</td>
<td>0.6</td>
</tr>
<tr>
<td>111</td>
<td>-1</td>
<td>64</td>
<td>0.7</td>
</tr>
</tbody>
</table>

The Binary Point

\[101.11_2 = 4 + 1 + \frac{1}{2} + \frac{1}{4} = 5.75\]

Observations:

- Divide by 2 by shifting point left
- \(0.111111\ldots_2\) is just below 1.0
- Some numbers cannot be exactly represented well
  \[1/10 \rightarrow 0.0001100110011[0011]^\ldots_2\]
### Obvious Approach: Fixed Point

\[ \sum_{k=-j}^{i} b_k \cdot 2^k \]

\[
\begin{array}{c}
b_i b_{i-1} \cdots b_2 b_1 b_0 b_{-1} b_{-2} b_{-3} b_{-j} \\
2^i \\
2^{i-1} \\
\cdots \\
4 \\
2 \\
1 \\
1/2 \\
1/4 \\
1/8 \\
2^{-j}
\end{array}
\]

### Fixed Point

**In** \( w \)-bits (\( w = i + j \)):
- use \( i \)-bits for left of binary point
- use \( j \)-bits for right of binary point

**Qualities:**
- Easy to understand
- Arithmetic relatively easy to implement…
- Precision and Magnitude:
  - 16-bits, \( i=j=8 \): 0 → 255.99609375
  - Step size: 0.00390625

### Another Approach: Scientific Notation

\[
6.02 \times 10^{23}
\]

**Sign, magnitude**

- **In Binary:**
  - \( \text{radix} = 2 \)
  - \( \text{value} = (-1)^s \times M \times 2^E \)

- How is this better than fixed point?
IEEE 754 Floating Point

- Established in 1980 as uniform standard for floating point arithmetic
- Supported by all major CPUs
- In 99.999% of all machines used today

Driven by Numerical Concerns
- Standards for rounding, overflow, underflow
- Primarily numerical analysts rather than hardware types defined standard

This is where it gets a little involved…

IEEE 754 Floating Point Standard

- Single precision: 8 bit exponent, 23 bit significand
- Double precision: 11 bit exponent, 52 bit significand

- Significand $M$ normally in range $[1.0,2.0) \Rightarrow$ Imply 1
- Exponent $E$ biased exponent $\Rightarrow$ B is bias ($B = 2^{E_{\text{bias}}-1} - 1$)

$$N = (-1)^s \times 1.M \times 2^{E-B}$$

- Bias allows integer comparison (almost)!
  - 0000…0000 is most negative exponent
  - 1111…1111 is most positive exponent

IEEE 754 Floating Point Example

Define Wimpy Precision as:
- 1 sign bit, 4 bit exponent, 3 bit significand, $B = 7$

Represent: -0.75

Represent as: $sE.M$
**IEEE 754 Floating Point**

**There's more!**

Normalized: $E \neq 000\ldots0$ and $E \neq 111\ldots1$

- Recall the implied $1.xxxxx$

Special Values: $E = 111\ldots1$

- $M = 000\ldots0$:
  - Represents +/- $\infty$ (infinity)
  - Used in overflow
  - Examples: $1.0/0.0 = +\infty$, $1.0/-0.0 = -\infty$
  - Further computations with infinity possible
  - Example: $X/0 > Y$ may be a valid comparison

**IEEE 754 Special Exponents**

Normalized: $E \neq 000\ldots0$ and $E \neq 111\ldots1$

Special Values: $E = 111\ldots1$

- $M \neq 000\ldots0$:
  - Not-a-Number (NaN)
  - Represents invalid numeric value or operation
  - Not a number, but not infinity (e.g. $\sqrt{-4}$)
  - Examples: $\sqrt{-1}$, $\infty - \infty$
  - NaNs propagate: $f(NaN) = NaN$

Denormalized: $E = 000\ldots0$

- $M = 000\ldots0$
  - Represents value 0
  - Note the distinct values +0 and −0
IEEE 754 Special Exponents

Normalized: \( E \neq 000\ldots0 \) and \( E \neq 111\ldots1 \)

- Recall the implied \( 1.xxxxx \)

Denormalized: \( E = 000\ldots0 \)

- \( M \neq 000\ldots0 \)
  - Numbers very close to 0.0
  - Lose precision as magnitude gets smaller
  - "Gradual underflow"

Exponent

\(-Bias + 1\)

Significand

\( 0.xxxx...x \)

Encoding Map

Dynamic Range

<table>
<thead>
<tr>
<th>S</th>
<th>E</th>
<th>M</th>
<th>exp</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Denormalized numbers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0000 000</td>
<td>n/a</td>
<td>0</td>
<td>closest to zero</td>
<td></td>
</tr>
<tr>
<td>0 0000 001</td>
<td>-6</td>
<td>1/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0000 010</td>
<td>-6</td>
<td>2/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0000 100</td>
<td>-6</td>
<td>4/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0000 110</td>
<td>-6</td>
<td>6/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0000 111</td>
<td>-6</td>
<td>8/512</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0001 000</td>
<td>-6</td>
<td>9/512</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Normalized numbers |
| 0 0110 110 | -1 | 28/32 |
| 0 0110 111 | -1 | 30/32 |
| 0 0111 000 | 0 | 1 |
| 0 0111 001 | 0 | 36/32 |
| 0 0111 010 | 0 | 40/32 |
| 0 1110 110 | 7 | 224 |
| 0 1110 111 | 7 | 240 |

- Largest norm

- Largest denorm

- Smallest norm

- Closest to 0 below

- Closest to 1 below

- Closest to 1 above
Define Wimpy Precision as:
1 sign bit, 4 bit exponent, 3 bit significand, $B = 7$

$E = 1-14$: Normalized
$E = 0$: Denormalized
$E = 15$: Infinity/ NaN