Binary Arithmetic
Binary Arithmetic

• Topics
  – Basic addition
  – Overflow
  – Multiplication & Division
Basic addition

- Binary and hex addition are fundamentally the same as decimal addition
  - Add digit-by-digit, using a carry as necessary
  - Result could require one more bit than the operands

\[
\begin{align*}
12540 & \quad \text{Dec} \quad 10011100 & \quad \text{Bin} \\
+ 4683 & \quad \text{Hex} \quad + 1010110
\end{align*}
\]

\[
\text{b0994f} \quad \text{Hex} \quad + \text{7120}
\]
Basic addition

- Binary and hex addition are fundamentally the same as decimal addition
  - Add digit-by-digit, using a carry as necessary
  - Result could require one more bit than the operands

11
12540
+ 4683
= 17223

111
10011100
+ 1010110
= 11110010

1
b0994f
+ 7120
b10a6f
Overflow

- Unsigned addition is subject to overflow
  - Caused by truncation to integer size

\[
\begin{array}{c}
994f \\
+ 7120 \\
\hline
10a6f = 0a6f
\end{array}
\]

(assume a 16-bit integer)

Figure 2.23: Unsigned addition. With a 4-bit word size, addition is performed modulo 16.
Two’s complement addition is identical to unsigned mechanically
- Subject to both positive and negative overflow
- Overflows if carry-in and carry-out differ for sign bit

Figure 2.24
Relation between integer and two’s-complement addition. When \( x + y \) is less than \(-2^{w-1}\), there is a negative overflow. When it is greater than or equal to \(2^{w-1}\), there is a positive overflow.

Figure 2.26
Two’s-complement addition. With a 4-bit word size, addition can have a negative overflow when \( x + y < -8 \) and a positive overflow when \( x + y \geq 8 \).

NOTE: this figure is printed incorrectly in your textbook!
Multiplication & Division

- Like addition, fundamentally the same as base 10
  - Actually, it’s even simpler!
  - Same regardless of encoding

- Special case: multiply by powers of 2 (shift left)

  \[
  \begin{align*}
  2 \ll 1 &= 4 & (2 \times 2) \\
  1 \ll 2 &= 4 & (1 \times 2 \times 2) \\
  1 \ll 4 &= 16 & (1 \times 2 \times 2 \times 2 \times 2) \\
  4 \ll 1 &= 8 & (4 \times 2) \\
  4 \ll 2 &= 16 & (4 \times 2 \times 2)
  \end{align*}
  \]

- Division is expensive!
  - Special case: divide by powers of two (shift right)
### One-byte integers:

<table>
<thead>
<tr>
<th>Binary</th>
<th>Unsigned</th>
<th>Two’s C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111 1111</td>
<td>255</td>
<td>-1</td>
</tr>
<tr>
<td>1111 1110</td>
<td>254</td>
<td>-2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1000 0001</td>
<td>129</td>
<td>-127</td>
</tr>
<tr>
<td>1000 0000</td>
<td>128</td>
<td>-128</td>
</tr>
<tr>
<td>0111 1111</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>0111 1110</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0000 0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0000 0000</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Overflow** when \(x + y > 255\)
- **Positive overflow** when \(x + y > 127\)
- **Negative overflow** when \(x + y < -128\)

**Figure 2.24**
Relation between integer and two’s-complement addition. When \(x + y\) is less than \(-2^{w-1}\), there is a negative overflow. When it is greater than or equal to \(2^{w-1}\), there is a positive overflow.
Now we can store integers

- But what about general real numbers?

Extend positional binary integers to store fractions

- Designate a certain number of bits for the fractional part
- These bits represent negative powers of two
- (Just like fractional digits in decimal fractions!)

\[
101.101 \\
\begin{array}{ccccccc}
4 & 2 & 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{8} \\
\hline \\
4 & 1 & 0.5 & 0.125 & & \\
\end{array}
\]

\[4 + 1 + 0.5 + 0.125 = 5.625 \]

(alternatively: \(5 + \frac{5}{8}\))
Another problem

- For scientific applications, we want to be able to store a wide range of values
  - From the scale of galaxies down to the scale of atoms
- Doing this with fixed-precision numbers is difficult
  - Even signed 64-bit integers
    - Perhaps allocate half for whole number, half for fraction
    - Range: ~$2 \times 10^{-9}$ through ~$2 \times 10^9$

Floating-point demonstration using Super Mario 64:
https://www.youtube.com/watch?v=9hdFG2GcNuA
Floating-point numbers

• Scientific notation to the rescue!
  – Traditionally, we write large (or small) numbers as \( x \cdot 10^e \)
  – This is how floating-point representations work
    • Store exponent and fractional parts (the significand) separately
    • The decimal point “floats” on the number line
    • Position of point is based on the exponent

\[
0.0123 \times 10^2 \\
0.123 \times 10^1 \\
1.23 = 1.23 \times 10^0 \\
12.3 \times 10^{-1} \\
123.0 \times 10^{-2}
\]
Floating-point numbers

• However, computers use binary
  – So floating-point numbers use base 2 scientific notation \((x \cdot 2^e)\)
• Fixed width field
  – Reserve one bit for the sign bit (0 is positive, 1 is negative)
  – Reserve \(n\) bits for biased exponent (bias is \(2^{n-1} - 1\))
    • Avoids having to use two’s complement
  – Use remaining bits for normalized fraction (implicit leading 1)
    • Exception: if the exponent is zero, don’t normalize

\[2.5 \rightarrow 0 1000 010\]

\[\text{Value} = (-1)^s \times 1.f \times 2^E\]