Principles Of Digital Design

Discussion: Arithmetic

Binary Arithmetic
Floating-Point Arithmetic
Binary Arithmetic

- Same basic methodology as decimal arithmetic
- Important to know number representation
  - Unsigned
  - Signed (signed-magnitude)
  - Two’s complement

- Binary values converted to decimal:

<table>
<thead>
<tr>
<th>Binary Number</th>
<th>Unsigned Value</th>
<th>Signed Value</th>
<th>Two's Complement Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>101101</td>
<td>45</td>
<td>-13</td>
<td>-19</td>
</tr>
<tr>
<td>011101</td>
<td>29</td>
<td>29</td>
<td>29</td>
</tr>
</tbody>
</table>
Binary Arithmetic: Unsigned

- Addition of unsigned binary numbers:
  - Valid range (6 bits): 0 – 63
  - Overflow for addition:
    - Number too large
    - Notify when overflow occurs

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_i$</td>
<td>$y_i$</td>
<td>$c_i$</td>
<td>$c_{i+1}$</td>
<td>$s_i$</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Addition of Binary Digits

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c}
\hline
32 & 16 & 8 & 4 & 2 & 1 \\
\hline
x & (45) & 1 & 0 & 1 & 1 & 0 & 1 \\
+ y & + (29) & 0 & 1 & 1 & 1 & 0 & 1 \\
Carries & & 1 & 1 & 1 & 1 & 0 & 1 \\
\hline
x + y & (74) & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
\]

Signed Binary Addition
Binary Arithmetic: Unsigned

- Subtraction of unsigned binary numbers:
  - Valid range (6 bits): 0 – 63
  - Overflow for subtraction:
    - Number too small
    - Notify when overflow occurs
**Binary Arithmetic: Signed-Magnitude**

- **Addition of signed-magnitude binary numbers:**
  - Valid range (6 bits): -31 – 31
  - Numbers are both positive:
    - *Sign change = overflow*
  - Numbers are both negative:
    - *Result is “+” & overflow: flip sign (result ok)*
    - *Result is “-” & overflow: overflow*
  - Numbers have different signs:
    - *No overflow*
    - **Methodology**
      - Larger number on top \((x)\)
      - Change sign of \(y\) and subtract

<table>
<thead>
<tr>
<th>+/−</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x) (29)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(+y) - (13)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Borrows**

| \(x + y\) (16) | 0 | 1 | 0 | 0 | 0 |

**Sign-Magnitude Binary Addition with Different Signs**

**Subtraction of Binary Digits**
Binary Arithmetic: Signed-Magnitude

- Subtraction of signed-magnitude numbers:
  - Valid range (6 bits): -31 – 31
  - Numbers have different signs:
    - Change sign of “-” and add
    - Overflow cases same as addition
  - Numbers are both positive:
    - Subtract normally
    - No overflow
  - Numbers are both negative:
    - Change sign of y and add
    - No overflow
  - Need to keep track of proper sign

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$c_i$</th>
<th>$c_{i+1}$</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Addition of Binary Digits

<table>
<thead>
<tr>
<th>$x$ (29)</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-y$ (13)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Carries</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$x - y$ (-10)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Sign-Magnitude Binary Subtraction with Different Signs
Binary Arithmetic: Two’s Complement

- Addition of two’s complement numbers:
  - Valid range (6 bits): -32 – 31
  - Numbers have different signs:
    - Can ignore carry (no overflow)
  - Numbers have same signs:
    - Sign change = overflow
  - Methodology:
    - Same as standard binary addition

Addition of Binary Digits

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$c_i$</th>
<th>$c_{i+1}$</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Two’s Complement Addition

<table>
<thead>
<tr>
<th>+/-</th>
<th>16</th>
<th>8</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ (-19)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$y$ + (29)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Carries</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$x + y$ (10)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Binary Arithmetic: Two’s Complement

- Subtraction of two’s complement numbers:
  - Valid range (6 bits): -32 – 31
  - Same overflow cases as two’s complement addition
  - Methodology:
    - Perform two’s complement on \( y \) and add

\[\begin{array}{c|ccccccc}
 x_i & y_i & c_i & c_{i+1} & s_i \\
\hline
 0  & 0  & 0  & 0  & 0  \\
 0  & 0  & 1  & 0  & 1  \\
 0  & 1  & 0  & 0  & 1  \\
 0  & 1  & 1  & 1  & 0  \\
 1  & 0  & 0  & 0  & 1  \\
 1  & 0  & 1  & 1  & 0  \\
 1  & 1  & 0  & 1  & 0  \\
 1  & 1  & 1  & 1  & 1  \\
\end{array}\]

Addition of Binary Digits

\[\begin{array}{c|ccccccc}
 x_i & y_i & c_i & c_{i+1} & s_i \\
\hline
 0  & 0  & 0  & 0  & 0  \\
 0  & 0  & 1  & 0  & 1  \\
 0  & 1  & 0  & 0  & 1  \\
 0  & 1  & 1  & 1  & 0  \\
 1  & 0  & 0  & 0  & 1  \\
 1  & 0  & 1  & 1  & 0  \\
 1  & 1  & 0  & 1  & 0  \\
 1  & 1  & 1  & 1  & 1  \\
\end{array}\]
Shift-and-add Multiplication

- Example of shift-and-add multiplication with unsigned binary numbers

\[
\begin{array}{c}
11110 \quad \text{multiplicand (30)} \\
\times \quad 101 \quad \text{multiplier (5)} \\
\hline
00000 \quad \text{initial partial product} \\
11110 \quad 1 \times \text{multiplicand, no shift} \\
\hline
11110 \quad \text{second partial product} \\
00000 \quad 0 \times \text{multiplicand, shift} \\
\hline
011110 \quad \text{third partial product} \\
11110 \quad 1 \times \text{multiplicand, shift} \\
1111 \quad \text{(carries)} \\
10010110 \quad \text{product (150)}
\end{array}
\]
Two’s-complement Multiplication

- Use multiplication procedure for unsigned numbers
- Extend partial products
- Negate multiplicand in last step if multiplier sign is negative

```
1 0 1 0  multiplicand (-6)
× 0 0 1 1  multiplier (3)
-----  extended partial product
0 0 0 0 0  1 * multiplicand, extend, no shift
1 1 0 1 0  1 * multiplicand, extend, shift
1 1 1 0 1 0  extended partial product
1 1 0 1 0
1 1 (carries)
1 1 0 1 1 1 0  extended partial product
1 1 1 0 1 1 1 0  extended partial product
0 0 0 0 0  0 * multiplicand, extend, shift
0 0 0 0 0  0 * multiplicand, extend, shift
1 1 1 0 1 1 1 0  product (-18)
```

Note:
- Red = ignored carry
Floating-Pointing Addition

- Problem: Add $1.110_2 \times 2^3$ and $1.011_2 \times 2^4$
- Procedure:
  1. Make two numbers have same exponents (shift mantissa)
     - Right shift 1.110 by 1 bit (divide by 2) to become 0.111
     - So $1.110 \times 2^3 = 0.111 \times 2^4$
  2. Add mantissas
     - $0.111 + 1.011 = 10.010$
  3. Normalize (shift mantissa)
     - Result: $10.010 \times 2^4 = 1.001 \times 2^5$