Lecture 3: Representing Signed Integers

CS 105
Memory: A (very large) array of bytes

- **Memory** is an array of **bits**

- A **byte** is a unit of eight bits

- An index into the array is an **address**, **location**, or **pointer**
  - Often expressed in hexadecimal

- We speak of the **value** in memory at an address
  - The value may be a single byte …
  - … or a multi-byte quantity starting at that address
Base-2 Integers (aka Binary Numbers)

128 (2^7)  64 (2^6)  32 (2^5)  16 (2^4)  8 (2^3)  4 (2^2)  2 (2^1)  1 (2^0)

0   0   0   0   0   0   1   0   1
0   0   1   0   0   1   1   1   1
1   1   1   1   1   1   1   1   1
Representing **Signed** Integers

- **Option 1: sign-magnitude**
  - One bit for sign; interpret rest as magnitude
Representing **Signed** Integers

- Option 2: one’s-complement
  - Complement all bits
Representing Signed Integers

- Option 3: two’s complement
  - Most used, like one’s complement, but with one zero value
  - Like unsigned, except the high-order contribution is negative
  - $Signed(x) = -x_{w-1} \cdot 2^{w-1} + \sum_{i=0}^{w-2} x_i \cdot 2^i$
Two’s Complement Signed Integers

• “Signed” does not mean “negative”

• High order bit is the *sign bit*
  • To negate, complement all the bits and add 1

• Arithmetic is the same as unsigned—same circuitry

• (Error conditions and comparisons are different)
Example: Three-bit integers

<table>
<thead>
<tr>
<th>unsigned</th>
<th>signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>−1</td>
<td>111</td>
</tr>
<tr>
<td>−2</td>
<td>110</td>
</tr>
<tr>
<td>−3</td>
<td>101</td>
</tr>
<tr>
<td>−4</td>
<td>100</td>
</tr>
</tbody>
</table>
Example: Three-bit integers

<table>
<thead>
<tr>
<th>unsigned</th>
<th>signed</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
</tr>
<tr>
<td>000</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>111</td>
</tr>
<tr>
<td>-2</td>
<td>110</td>
</tr>
<tr>
<td>-3</td>
<td>101</td>
</tr>
<tr>
<td>-4</td>
<td>100</td>
</tr>
</tbody>
</table>

- The high-order bit is the *sign bit*.
- The largest unsigned value is 11...1, $\text{UMax}$.
- The signed value for –1 is always 11...1.
- Signed values range between $\text{TMin}$ and $\text{TMax}$.

This representation of signed values is called *two’s complement*.
## Important Signed Numbers

<table>
<thead>
<tr>
<th>w</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>0x7F</td>
<td>0x7FFF</td>
<td>0x7FFFFFFFF</td>
<td>0x7FFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>Min</td>
<td>0x80</td>
<td>0x8000</td>
<td>0x80000000</td>
<td>0x80000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>0x00</td>
<td>0x0000</td>
<td>0x00000000</td>
<td>0x000000000000000000</td>
</tr>
<tr>
<td>-1</td>
<td>0xFF</td>
<td>0xFFFF</td>
<td>0xFFFFFFFF</td>
<td>0xFFFFFFFFFFFFFFFF</td>
</tr>
</tbody>
</table>
Exercise 1: Signed Integers

Assume an 8 bit (1 byte) signed integer representation using two’s complement:

• What is the binary representation for 47? 00101111

• What is the binary representation for -47? 11010001

• What is the number represented by 10000110? -122

• What is the number represented by 00100101? 37
Casting between Numeric Types

- Casting from shorter to longer types preserves the value

- Casting from longer to shorter types drops the high-order bits (modulus)

- Casting between signed/unsigned types preserves the bits (it just changes the interpretation)

- Implicit casting occurs in assignments and parameter lists. In mixed expressions, signed values are implicitly cast to unsigned
  - Source of many errors!
Exercise 2: Casting

- Assume you have a machine with 6-bit integers/3-bit shorts
- Assume variables: `int x = -17; short sy = -3;`
- Complete the following table

<table>
<thead>
<tr>
<th>Expression</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-17</td>
<td></td>
</tr>
<tr>
<td>sy</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>(unsigned) x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(int) sy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(short) x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 2: Casting

- Assume you have a machine with 6-bit integers/3-bit shorts
- Assume variables: `int x = -17; short sy = -3;`
- Complete the following table

<table>
<thead>
<tr>
<th>Expression</th>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>-17</td>
<td>101111</td>
</tr>
<tr>
<td>sy</td>
<td>-3</td>
<td>101</td>
</tr>
<tr>
<td>(unsigned) x</td>
<td>47</td>
<td>101111</td>
</tr>
<tr>
<td>(int) sy</td>
<td>-3</td>
<td>111101</td>
</tr>
<tr>
<td>(short) x</td>
<td>-1</td>
<td>111</td>
</tr>
</tbody>
</table>
When to Use Unsigned

- Rarely
- When doing multi-precision arithmetic, or when you need an extra bit of range … but be careful!

```c
unsigned i;
for (i = cnt-2; i >= 0; i--){
    a[i] += a[i+1];
}
```
Arithmetic Logic Unit (ALU)

- A circuit that performs bitwise operations and arithmetic on integer binary types

Status examples:
- Carry-out
- Zero
- Negative
- Overflow
- parity

Opcode examples:
- Add, Subtract
- Increment, Decrement
- AND, OR, XOR
- Shift, Rotate
Bitwise vs Logical Operations in C

• **Bitwise Operators** &, |, ~, ^
  - View arguments as bit vectors
  - Operations applied bit-wise in parallel

• **Logical Operators** &&, ||, !
  - View 0 as “False”
  - View anything nonzero as “True”
  - Always return 0 or 1
  - **Short-circuit termination**

• **Shift operators** <<, >>
  - Left shift fills with zeros
  - For signed integers, **right shift is arithmetic** (fills with high-order bit)
Exercise 3: Bitwise vs Logical Operations

• Assume signed char data type (one byte)

  • \(\sim (-30)\) \quad = \sim11100010 = 00011101 = 29
  • \(!(-30)\) \quad = !11100010 = 00000000 = 0
  • 120 & 85 \quad = 01111000 \& 01010101 = 01010000 = 80
  • 120 | 85 \quad = 01111000 | 01010101 = 01111101 = 125
  • 120 && 85 \quad = 01111000 \&\& 01010101 = 00000001 = 1
  • 120 || 85 \quad = 01111000 || 01010101 = 00000001 = 1

  • \(-106 \ll 4\) \quad = 10010110 \ll 4 = 01100000 = 96
  • \(-106 \ll 2\) \quad = 10010110 \ll 2 = 01011000 = 88
  • \(-106 \gg 4\) \quad = 10010110 \gg 4 = 11111001 = -7
  • \(-106 \gg 2\) \quad = 10010110 \gg 2 = 11100101 = -27
Addition/Subtraction Example

• Compute $5 + (-3)$ assuming all ints are stored as four-bit signed values

\[
\begin{array}{c}
1 \\
0101 \\
1101 \\
\hline
0010
\end{array}
\]

$0010 = 2$ (Base-10)

Exactly the same as unsigned numbers!

… but with different error cases
Addition/Subtraction with Overflow

- Compute $5 + 3$ assuming all ints are stored as **four-bit signed** values

\[
\begin{array}{c}
111 \\
0101 \\
+ 0011 \\
\hline
1000
\end{array} = -8 \text{ (Base-10)}
\]
Error Cases

• Assume $w$-bit signed values

\[
\begin{align*}
-2 \cdot 2^{w-1} & \quad -2^{w-1} & \quad 0 & \quad 2^{w-1} & \quad 2 \cdot 2^{w-1} \\
\text{representable values} & & & & \\
\text{Possible values of } x + y & & & & \\
\end{align*}
\]

\[
\begin{align*}
x + y - 2^w & \quad \text{(positive overflow)} \\
x + y & \quad \text{(normal)} \\
x + y + 2^w & \quad \text{(negative overflow)}
\end{align*}
\]

• overflow has occurred iff $x > 0$ and $y > 0$ and $x + t_w y < 0$
  or $x < 0$ and $y < 0$ and $x + t_w y > 0$
Exercise 4: Binary Addition

• Given the following 5-bit signed values, compute their sum and indicate whether an overflow occurred.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x+y</td>
<td>overflow?</td>
</tr>
<tr>
<td>00010</td>
<td>00101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01100</td>
<td>00100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>10001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 4: Binary Addition

- Given the following 5-bit signed values, compute their sum and indicate whether an overflow occurred.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x+y</td>
<td>overflow?</td>
<td></td>
</tr>
<tr>
<td>00010</td>
<td>00101</td>
<td>00111</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>01100</td>
<td>00100</td>
<td>10000</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>10100</td>
<td>10001</td>
<td>00101</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
Multiplication Example

• Compute $3 \times 2$ assuming all ints are stored as four-bit signed values

\[
\begin{array}{cccc}
0 & 0 & 1 & 1 \\
\times & 0 & 0 & 1 & 0 \\
\hline
0 & 0 & 0 & 0 \\
+ & 0 & 0 & 1 & 1 & 0 \\
\hline
0 & 1 & 1 & 0
\end{array}
\]

= 6 (Base-10)

Exactly like unsigned multiplication!

… except with different error cases
Multiplication Example

- Compute 5 x 2 assuming all ints are stored as four-bit signed values

\[
\begin{array}{c}
0101 \\
\times 0010 \\
\hline
0000 \\
+010110 \\
\hline
1010
\end{array}
\]

= -6 (Base-10)
Error Cases

- Assume $w$-bit unsigned values

- $x \cdot y = U2T((x \cdot y) \mod 2^w)$
Exercise 5: Binary Multiplication

Given the following 3-bit signed values, compute their product and indicate whether an overflow occurred.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x*y</th>
<th>overflow?</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 5: Binary Multiplication

- Given the following 3-bit signed values, compute their product and indicate whether an overflow occurred.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>x*y</td>
<td>overflow?</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>100</td>
<td>yes</td>
</tr>
<tr>
<td>010</td>
<td>011</td>
<td>110</td>
<td>yes</td>
</tr>
<tr>
<td>111</td>
<td>010</td>
<td>110</td>
<td>no</td>
</tr>
</tbody>
</table>