Review: Unsigned Integers
Weighted positional notation
- like decimal numbers: “329”
- “3” is worth 300, because of its position, while “9” is only worth 9

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

Review: Signed Integers
With n bits, we have 2^n distinct values.
- assign about half to positive integers (1 through 2^n-1)
- that leaves two values: one for 0, and one extra

Positive integers
- just like unsigned – zero in most significant bit

Negative integers
- sign-magnitude – set top bit to show negative, other bits are the same as unsigned
- one’s complement – flip every bit to represent negative
- in either case, MS bit indicates sign: 0=positive, 1=negative
Two's Complement

Problems with sign-magnitude and 1's complement
- two representations of zero (+0 and –0)
- arithmetic circuits are complex
  - Adding a negative number ⇒ subtraction
  - Need to “correct” result to account for borrowing

Two’s complement representation developed to make circuits easy for arithmetic.
- for each positive number (X), assign value to its negative (-X), such that X + (-X) = 0 with “normal” addition, ignoring carry out

Two’s Complement Representation
If number is positive or zero,
- normal binary representation, zeroes in upper bit(s)
If number is negative,
- start with positive number
- flip every bit (i.e., take the one’s complement)
- then add one

Two’s Complement Signed Integers
MS bit is sign bit – it has weight \(-2^{n-1}\).
Range of an n-bit number: \(-2^{n-1}\) through \(2^{n-1} - 1\).
- The most negative number \((-2^{n-1})\) has no positive counterpart.

Converting 2’s Complement to Decimal
1. If leading bit is one, take two's complement to get a positive number.
2. Add powers of 2 that have “1” in the corresponding bit positions.
3. If original number was negative, add a minus sign.

Converting Decimal to Binary (2’s C)
1. Change to positive decimal number.
2. Use either repeated division by 2 or repeated subtraction of powers of two
3. Append a zero as MS bit; if original was negative, take two’s complement.
Operations: Arithmetic and Logical
Recall:
a data type includes representation and operations.
We now have a good representation for signed integers, so let’s look at some arithmetic operations:
• Addition
• Subtraction
• Sign Extension
(We’ll also look at overflow conditions for addition.)
Multiplication, division, etc., can be built from these basic operations.
Review: Logical operations are also useful:
• AND
• OR
• NOT

Addition
As we’ve discussed, 2’s comp. addition is just binary addition.
• assume all integers have the same number of bits
• ignore carry out
• for now, assume that sum fits in n-bit 2’s comp. representation

\[
\begin{align*}
01101000 \; (104) & \quad + \quad 1110000 \; (-16) & \quad & = \quad 11111010 \; (-9) \\
01011000 \; (98) & \quad + \quad 11110011 \; (-9) & \quad & = \quad 11110111 \; (-19)
\end{align*}
\]
Assuming 8-bit 2’s complement numbers.

Subtraction
Negate subtrahend (2nd no.) and add.
• assume all integers have the same number of bits
• ignore carry out
• for now, assume that difference fits in n-bit 2’s comp. representation

\[
\begin{align*}
01101000 \; (104) & \quad - \quad 00010000 \; (16) & \quad & = \quad 01100100 \; (88) \\
01101000 \; (104) & \quad + \quad 11110011 \; (-9) & \quad & = \quad 01011000 \; (98)
\end{align*}
\]
Assuming 8-bit 2’s complement numbers.

Sign Extension
To add two numbers, we must represent them with the same number of bits.
If we just pad with zeroes on the left:

\[
\begin{align*}
\text{4-bit} & \quad \text{8-bit} \\
0100 \; (4) & \quad 00000100 \; (still \; 4) \\
1100 \; (-4) & \quad 00001100 \; (12, \; not \; -4)
\end{align*}
\]
Instead, replicate the MS bit -- the sign bit:

\[
\begin{align*}
\text{4-bit} & \quad \text{8-bit} \\
0100 \; (4) & \quad 00000100 \; (still \; 4) \\
1100 \; (-4) & \quad 11111100 \; (still \; -4)
\end{align*}
\]

Overflow
If operands are too big, then sum cannot be represented as an n-bit 2’s comp number.

\[
\begin{align*}
01000 \; (8) & \quad + \quad 11000 \; (-8) \\
01001 \; (9) & \quad + \quad 10111 \; (-9) \\
10001 \; (-15) & \quad + \quad 01111 \; (+15)
\end{align*}
\]
We have overflow if:
• signs of both operands are the same, and
• sign of sum is different.
Another test -- easy for hardware:
• carry into MS bit does not equal carry out

Fractions: Fixed-Point
How can we represent fractions?
• Use a “binary point” to separate positive from negative powers of two -- just like “decimal point.”
• 2’s comp addition and subtraction still work.
  > if binary points are aligned
    \[
    \begin{align*}
    2^1 \cdot 0.5 & = 0.5 \\
    2^1 \cdot 0.25 & = 0.25 \\
    2^2 \cdot 0.125 & = 0.25
    \end{align*}
    \]
\[
\begin{align*}
00101000.101 & + \quad 01111110.110 & = \quad 01010111.011
\end{align*}
\]
No new operations -- same as integer arithmetic.
Very Large and Very Small: Floating-Point

Large values: \(6.023 \times 10^{23}\) -- requires 79 bits
Small values: \(6.626 \times 10^{-34}\) -- requires >110 bits

Use equivalent of “scientific notation”: \(F \times 2^E\)

Need to represent \(F\) (fraction), \(E\) (exponent), and sign.

IEEE 754 Floating-Point Standard (32-bits):

<table>
<thead>
<tr>
<th>Exponent</th>
<th>Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

\[N = -1^S \times 1.\text{fraction} \times 2^{\text{exponent}-127}, \quad 1 \leq \text{exponent} \leq 254\]
\[N = -1^S \times 0.\text{fraction} \times 2^{-126}, \quad \text{exponent} = 0\]

Floating Point Example

Single-precision IEEE floating point number:

\[10111111101000000000000000000000\]

- Sign is 1 – number is negative.
- Exponent field is 01111110 = 126 (decimal).
- Fraction is 0.100000000000… = 0.5 (decimal).

Value = \(-1.5 \times 2^{(126-127)} = -1.5 \times 2^{-1} = -0.75\).

LC-3 Data Types

Some data types are supported directly by the instruction set architecture.

For LC-3, there is only one supported data type:
- 16-bit 2’s compliment signed integer
- Operations: ADD, AND, NOT

Other data types are supported by interpreting 16-bit values as logical, text, fixed-point, etc., in the software that we write.

Summary

Review: unsigned numbers
New: signed numbers
  - Sign/magnitude and one’s complement
  - Two’s complement
Two’s Complement operations & issues
  - Addition, subtraction
  - Sign extension
  - Overflow
Fractions
  - Fixed point
  - Floating point: IEEE754 standard