What About Negative Numbers?

Last time, we developed
◦ the $N$-bit unsigned representation
◦ for integers in the range $[0, 2^N - 1]$

Now, let’s think about negative numbers.
◦ How should we represent them?
◦ Can we use a minus sign?

$-11000 = -24_{10}$?

There’s no “−” in a bit!

One Option: The Signed-Magnitude Representation

But we can use another bit for a sign:

$0 \rightarrow +$, and $1 \rightarrow −$

Doing so gives the $N$-bit signed-magnitude representation:

- **Sign**
- *(N-1)-bit magnitude*

This representation can represent numbers in the range $[-(2^{N-1} - 1), 2^{N-1} - 1]$.

What Happened to the Last Bit Pattern?

Signed-magnitude was used in some early computers (such as the IBM 704 in 1954).

A question for you:
◦ The range represented is $[-(2^{N-1} - 1), 2^{N-1} - 1]$.
◦ That gives $2^N - 1$ different numbers.
◦ What’s the last pattern being used to represent?
Signed-Magnitude Has Two Patterns for Zero

There are two bit patterns for 0!

<table>
<thead>
<tr>
<th>0</th>
<th>0000...0000</th>
<th>+0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000...0000</td>
<td>-0</td>
</tr>
</tbody>
</table>

This aspect made some hardware more complex than is necessary.

Modern machines do not use signed-magnitude.

How Do We Choose Among Representations?

What makes a representation good?

- efficient: most bit patterns represent some item uniquely (so, not unary!)
- easy/fast implementation of common operations: such as arithmetic for numbers
- shared implementation with other representations: in this case, implementation is "free" in some sense

Representations Can be Chosen to Share Hardware

Imagine a device that performs addition on two bit patterns of an unsigned representation.

<table>
<thead>
<tr>
<th>“2”</th>
<th>0</th>
<th>unsigned</th>
<th>0</th>
<th>“5”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“3”</td>
<td>0</td>
<td>adder</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Can we use the same “adder” device for signed numbers? Yes! If we choose the right representations.

Add Unsigned Bit Patterns Using Base 2 Addition

Recall that the unsigned representation is drawn from base 2.

We use base 2 addition for unsigned patterns.

- Like base 10, we add digit by digit.
- Unlike base 10, the single-digit table of sums is quite small...
- What is $1 + 1 + 1$?
Example: Addition of Unsigned Bit Patterns

Let's do an example with 5-bit unsigned

\[
\begin{array}{c}
11 \\
01110 \quad (14) \\
+ \quad 00100 \quad (4) \\
\hline
10010 \quad (18)
\end{array}
\]

Good, we got the right answer!

Overflow Can Occur with Unsigned Addition

The unsigned representation is fixed width.  
• If we start with \( N \) bits,  
• we must end with \( N \) bits.

What is the condition under which the sum cannot be represented?  
• The sum should have a 1 in the \( 2^N \) place.  
• Only occurs when the most significant bits of the addends generate a carry.

We call this condition an overflow.

Example: Overflow of Unsigned Bit Patterns

Let's do another example, again with 5-bit unsigned

\[
\begin{array}{c}
01110 \quad (14) \\
+ \quad 10101 \quad (21) \\
\hline
00011 \quad (3)
\end{array}
\]

Oops! (The carry out indicates an overflow for unsigned addition.)

Unsigned Addition is Modular Arithmetic

Modular arithmetic is related to the idea of the “remainder” of a division.

Given integers \( A, B, \) and \( M, \)
• \( A \) and \( B \) are said to be equal mod \( M \) iff*
• \( A = B + kM \) for some integer \( k. \)

Note that \( k \) can be negative or zero, too.

We write: \((A = B)\mod M.\)

* “iff” means “if and only if,” an implication in both directions, and is often used for mathematical definitions.
Unsigned Addition is Always Correct Mod $2^N$

Let $SUM_N(A,B)$ be the number represented by the sum of two $N$-bit unsigned bit patterns.
If no overflow occurs ($A + B < 2^N$), we have $SUM_N(A,B) = A + B$.
For sums that produce an overflow, the bit pattern of the sum is missing the $2^N$ bit, so $SUM_N(A,B) = A + B - 2^N$.
In both cases, $(SUM_N(A,B) = A + B) \mod 2^N$.

Modular Arithmetic Key to Good Integer Representations

Modular arithmetic is the key.
It allows us to define
- a representation for signed integers
- that uses the same devices
- as are needed for unsigned arithmetic.
The representation is called 2’s complement.
Details soon…

Modular Arithmetic on the Number Line

To understand modular arithmetic graphically, imagine breaking the number line into groups of $M$ numbers, as shown above for $M=8$.
Two numbers are equal mod $M$ if they occupy the same position in their respective groups.
For example, 0 is equal to an infinite number of other numbers (... -24, -16, -8, 8, 16, 24, ...).
We usually name sets of numbers that are equal mod $M$ using the number in the range $[0, M-1]$. 