Integer Representation

Representing Negative Numbers
Signed Magnitude
Two’s Complement
Arithmetic
Shifting with signed numbers
sign-magnitude

Most-significant bit (MSB) is sign bit
  - 0 means non-negative
  - 1 means negative

Remaining bits are an unsigned magnitude

8-bit sign-magnitude:
  - 00000000 represents _____
  - 01111111 represents _____
  - 10000101 represents _____
  - 10000000 represents _____

Anything weird here?

Arithmetic?

Example:
4 - 3 != 4 + (-3)

Zero?
(4-bit) two's complement signed integer representation

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
-2^3 & 2^2 & 2^1 & 2^0
\end{array}
\]

\[= 1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0\]

4-bit two's complement integers:

minimum =

maximum =
## two's complement vs. unsigned

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[2^{n-1}]</td>
<td>[2^{n-2}]</td>
<td>[2^2]</td>
<td>[2^1]</td>
<td>[2^0]</td>
</tr>
<tr>
<td>[-2^{n-1}]</td>
<td>[2^{n-2}]</td>
<td>[2^2]</td>
<td>[2^1]</td>
<td>[2^0]</td>
</tr>
</tbody>
</table>

What's the difference?

\[n\text{-bit minimum} = 2^n - 1\]
\[n\text{-bit maximum} = 2^n - 1\]
8-bit representations

00001001 100000001

111111111 00100111
4-bit unsigned vs. 4-bit two's complement

1011

1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0

1 \times -2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0

11

-5

4-bit unsigned

4-bit two's complement
two's complement addition

\[
\begin{array}{cccc}
2 & 0010 & -2 & 1110 \\
+ 3 & + 0011 & + -3 & + 1101 \\
\end{array}
\]

\[
\begin{array}{cccc}
-2 & 1110 & 2 & 0010 \\
+ 3 & + 0011 & + -3 & + 1101 \\
\end{array}
\]
two’s complement overflow

Addition overflows

-1 1111
+ 2 + 0010
——— ———
6 0110
+ 3 + 0011

Modular Arithmetic

Some CPUs/languages raise exceptions on overflow. C and Java cruise along silently... Feature? Oops?
Exploded due to *cast* of 64-bit floating-point number to 16-bit signed number. Overflow.

"... a Model 787 airplane ... can lose all alternating current (AC) electrical power ... caused by a *software counter* internal to the GCUs that will overflow after 248 days of continuous power. We are issuing this AD to prevent loss of all AC electrical power, which could result in loss of control of the airplane." --FAA, April 2015
A few reasons two’s complement is awesome

- Addition, subtraction, hardware
- Sign
- Negative one
- Complement rules
Convert/cast signed number to larger type.

0 0 0 0 0 0 1 0 8-bit 2

__________ 0 0 0 0 0 0 1 0 16-bit 2

1 1 1 1 1 1 0 0 8-bit -4

__________ 1 1 1 1 1 1 0 0 16-bit -4

Rule/name?
unsigned shifting and arithmetic

unsigned

\[
x = 27; \quad 0 0 0 1 1 0 1 1
\]

\[
y = x \ll 2; \quad \text{logical shift left}
\]

\[
y == 108 \quad 0 0 0 1 1 0 1 1 0 0
\]

\[
x = 237; \quad 1 1 1 0 1 1 0 1 1 0 1
\]

\[
y = x \gg 2; \quad \text{logical shift right}
\]

\[
y == 59 \quad 0 0 1 1 1 0 1 1 0 1 0 1
\]
two's complement shifting and arithmetic

**Signed**

\[ x = -101; \]

\[ y = x << 2; \]

\[ y == 108 \]

\[ x*2 \mod 2 \]

**Arithmetic shift right**

\[ x = -19; \]

\[ y = x >> 2; \]

\[ y == -5 \]

**Logical shift left**

\[ y = x >> 2; \]
shift-and-add

Available operations
- $x \ll k$ implements $x \times 2^k$
- $x + y$

Implement $y = x \times 24$ using only $\ll$, $+$, and integer literals
Casting Integers in C

Number literals: 37 is signed, 37U is unsigned

Integer Casting: *bits unchanged, just reinterpreted.*

Explicit casting:

```c
int tx = (int) 73U;          // still 73
unsigned uy = (unsigned) -4; // big positive #
```

Implicit casting:

```c
tx = ux;                     // tx = (int)ux;
uy = ty;                     // uy = (unsigned)ty;
void foo(int z) { ... }
foo(ux);                     // foo((int)ux);
if (tx < ux) ...             // if ((unsigned)tx < ux) ...
```

Actually does
If you mix unsigned and signed in a single expression, then signed values are implicitly cast to unsigned.

<table>
<thead>
<tr>
<th>Argument₁</th>
<th>Op</th>
<th>Argument₂</th>
<th>Type</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>==</td>
<td>0U</td>
<td>unsigned</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>&lt;</td>
<td>0</td>
<td>signed</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>&lt;</td>
<td>0U</td>
<td>unsigned</td>
<td>0</td>
</tr>
<tr>
<td>2147483647</td>
<td>&lt;</td>
<td>-2147483648</td>
<td>unsigned</td>
<td>0</td>
</tr>
<tr>
<td>2147483647U</td>
<td>&lt;</td>
<td>-2147483648</td>
<td>unsigned</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>&lt;</td>
<td>-2</td>
<td>(int)</td>
<td>0</td>
</tr>
<tr>
<td>(unsigned)-1</td>
<td>&lt;</td>
<td>-2</td>
<td>(int)</td>
<td>0</td>
</tr>
<tr>
<td>2147483647</td>
<td>&lt;</td>
<td>2147483648U</td>
<td>(int)</td>
<td>0</td>
</tr>
<tr>
<td>2147483647</td>
<td>&lt;</td>
<td>(int)2147483648U</td>
<td>(int)</td>
<td>0</td>
</tr>
</tbody>
</table>

Note: \( T_{min} = -2,147,483,648 \quad T_{max} = 2,147,483,647 \)