(4-bit) unsigned integer representation

\[
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
8 & 4 & 2 & 1 \\
2^3 & 2^2 & 2^1 & 2^0 \\
3 & 2 & 1 & 0
\end{array}
\]

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

\[\text{weight} \quad \text{position}\]

n-bit unsigned integers:

minimum =

maximum =

 modular arithmetic, overflow

\[
\begin{array}{c}
11 \quad 1011 \\
+2 \quad +0010 \\
\end{array}
\]

\[
\begin{array}{c}
13 \quad 1101 \\
+5 \quad +0101 \\
\end{array}
\]

\[
\begin{array}{ccccc}
15 & 1111 & 0000 & 0 & 1 \\
14 & 1110 & 0001 & 0 & 0 \\
13 & 1101 & 0010 & 0 & 1 \\
12 & 1100 & 0011 & 0 & 1 \\
11 & 1011 & 0100 & 1 & 1 \\
10 & 1010 & 0101 & 1 & 0 \\
9 & 1001 & 0110 & 0 & 1 \\
8 & 1000 & 0111 & 0 & 0 \\
7 & 0111 & 1000 & 0 & 0 \\
6 & 0110 & 1001 & 0 & 0 \\
5 & 0101 & 1010 & 0 & 0 \\
4 & 0100 & 1011 & 0 & 0 \\
3 & 0011 & 1100 & 0 & 0 \\
2 & 0010 & 1101 & 0 & 0 \\
1 & 0001 & 1110 & 0 & 0 \\
0 & 0000 & 1111 & 0 & 0
\end{array}
\]

\[\text{x+y} \quad \text{in} \quad n\text{-bit unsigned arithmetic is} \quad \text{in math} \]

\[\text{unsigned overflow} = \quad = \]

Unsigned addition overflows if and only if

(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

\[
\begin{array}{cccc}
00000100 \\
+10000011
\end{array}
\]

Anything weird here?

\text{Arithmetic?}

\text{Example:}

\[4 - 3 != 4 + (-3)\]

8-bit sign-magnitude:

00000000 represents _____

01111111 represents _____

10000101 represents _____

10000000 represents _____

Anything weird here?

Ex

4-bit two's complement integers:

minimum =

maximum =
two’s complement vs. unsigned

```
<table>
<thead>
<tr>
<th>2^{n-1}</th>
<th>2^{n-2}</th>
<th>...</th>
<th>2^2</th>
<th>2^1</th>
<th>2^0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2^{n-1}</td>
<td>-2^{n-2}</td>
<td>...</td>
<td>-2^2</td>
<td>-2^1</td>
<td>-2^0</td>
</tr>
</tbody>
</table>
```

What's the difference?

n-bit minimum = n-bit maximum =

8-bit representations

```
0 0 0 0 1 0 0 1     1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1     0 0 1 0 0 1 1 1
```

n-bit two's complement numbers:

minimum = maximum =

two’s complement addition

```
+ 3   + 0011   + -3   + 1101
2     0010    -2    1110
```

-2    1110    2     0010
+ 3   + 0011   + -3   + 1101

Modular Arithmetic
two’s complement **overflow**

**Addition overflow**
- if and only if
- if and only if

1. \( -1 = 111 \)
2. \( +2 = +0010 \)
3. \( 6 = 0110 \)
4. \( +3 = +0011 \)

---

**Modular Arithmetic**

Some CPUs/languages raise exceptions on overflow.
C and Java cruise along silently... Feature? Oops?

---

**A few reasons two’s complement is awesome**

Addition, subtraction, hardware

- Sign
- Negative one
- Complement rules

---

**Another derivation**

**How should we represent 8-bit negatives?**

- For all positive integers \( x \),
  we want the representations of \( x \) and \( -x \) to sum to zero.
- We want to use the standard addition algorithm.

\[
\begin{array}{c}
00000001 \\
+ \quad 0000010 \\
\hline
00000000 \\
\end{array}
\begin{array}{c}
00000001 \\
+ \quad 0000010 \\
\hline
00000000 \\
\end{array}
\begin{array}{c}
00000001 \\
+ \quad 0000010 \\
\hline
00000000 \\
\end{array}
\]

- Find a rule to represent \( -x \) where that works...

---

**Convert/cast signed number to larger type.**

\[
\begin{array}{c}
\text{0 0 0 0 0 0 1 0} \quad \text{8-bit 2} \\
\hline
\text{1 1 1 1 1 1 0 0} \quad \text{8-bit -4} \\
\hline
\text{0 0 0 0 0 0 1 0} \quad \text{16-bit 2} \\
\hline
\text{1 1 1 1 1 1 0 0} \quad \text{16-bit -4} \\
\end{array}
\]

Rule/name?
unsigned shifting and arithmetic

unsigned x = 27;
0 0 0 1 1 0 1 1
y = x << 2;
y == 108 0 0 0 1 1 0 1 1 0 0

unsigned x = 237;
1 1 1 0 1 1 0 1
y = x >> 2;
y == 59
0 0 1 1 1 0 1 1 0 1

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

two's complement shifting and arithmetic

signed x = -101;
1 0 0 1 1 0 1 1
y = x << 2;
y == 108 1 0 0 1 1 0 1 1 0 0

signed x = -19;
1 1 1 0 1 1 0 1
y = x >> 2;
y == -5 1 1 1 1 0 1 1 0 1 1 0 1

What does this function compute?

unsigned puzzle(unsigned x, unsigned y) {
    unsigned result = 0;
    for (unsigned i = 0; i < 32; i++){
        if (y & (1 << i)) {
            result = result + (x << i);
        }
    }
    return result;
}