N-bit <u>unsigned</u> integers

000...0   111...1

$\emptyset$ (smallest) 2^N-1 (largest)

N-bit <u>signed</u> integer

- need to represent negative numbers
  - sign+magnitude is <u>not</u> used.
  - <u>two's complement</u> is used.

What does "$-x$" mean?

- it is the additive inverse of x

$$x + -x = 0$$

We'll use 4-bit numbers as examples

- computers have 32 or 64-bit numbers

4-bit addition:

$$
\begin{array}{r}
0110 \\
1111 \\
\hline
\underset{\text{carry}}{1}\;\underbrace{0101}_{\text{result}}
\end{array}
$$

If the result can only be represented as 4 bits, the carry is discarded.

What can we add to 1 to get 0, using only 4-bit numbers?

$$
\begin{array}{r}
1 \\
-1 \\
\hline
0
\end{array}
\qquad\qquad
\begin{array}{r}
0001 \\
1111 \;\leftarrow -1 \\
\hline
1\;\underbrace{0000}_{\text{result}=0}
\end{array}
$$

How to represent the rest of the negative numbers?

- hint: for $-x$, pick a number that when you add it to x you get all ones, $111\ldots1$, and then add 1.
  - that will give us all 0's (and a carry to be discarded).

What can we add to x to get all 1's ($11\ldots1$)?

- Answer: the complement of x (flip each bit)

$$
\begin{array}{lrl}
x & 1011 & \\
\sim x & 0100 & \leftarrow \text{complement of } x \text{ (flip each bit)} \\
\hline
 & 1111 & \\
\text{Then add 1!} & 0001 & \leftarrow \text{add 1} \\
\hline
 & 1\ \underbrace{0000}_{0} &
\end{array}
$$

So: $x + \sim x + 1 = 0$

complement of x.

Because addition is associative:

$$x + \underbrace{(\sim x + 1)} = 0$$

So, for any x, the value of $-x$ is computed by flipping the bits of x and adding 1.

Example: 10 decimal = 1010 binary

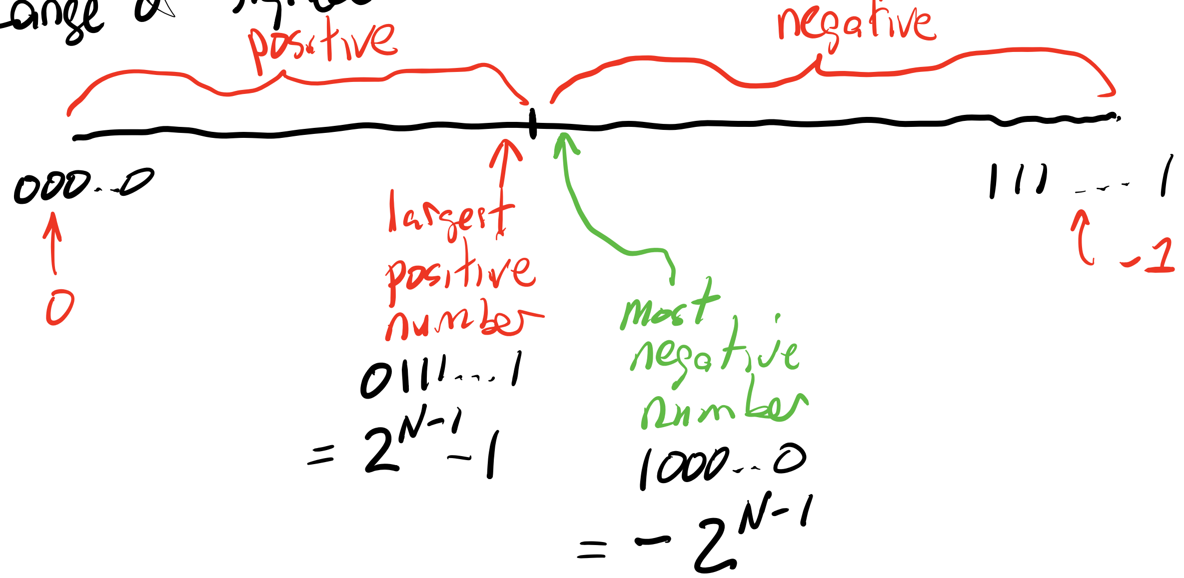
Flipping the bits (0101) and adding 1:

$$\begin{array}{r} \sim x \\ 0101 \\ 1 \\ \hline 0110 \\ -x \end{array}$$

So 0110 in binary is −10 decimal:

$$\begin{array}{rl} 1010 & \leftarrow 10 \text{ decimal} \\ 0110 & \leftarrow -10 \text{ decimal} \\ \hline 0000 & \end{array}$$

Range of signed numbers: (N bits)



You can recognize a negative number because its <u>leftmost bit</u> will be 1.

- so, the leftmost bit indicates the sign of the number.
  - still not sign + magnitude, because the rest of the bits of a negative number do not give the magnitude (i.e. the distance from zero).

The value of an N-bit unsigned number

$$b_{n-1}\, b_{n-2} \cdots b_1\, b_0$$

leftmost bit

rightmost bit

is $$\sum_{i=0}^{n-1} \left(b_i \times 2^i \right)$$

this represents the "column" ("1's column", "2's column", "4's column", etc.)

The value of an N-bit two's complement (signed) number

$$b_{n-1}\, b_{n-2} \cdots b_1\, b_0$$

most negative number

is

$$\left(b_{n-1} \times -2^{n-1} \right) + \sum_{i=0}^{n-2} \left(b_i \times 2^i \right)$$

leftmost bit

In C, the logical operators are

&& : AND
|| : OR
! : NOT

These operators treat their operand as a single boolean value

00...0 - zero means false
- non-zero means true

So,

x && y : produces a non-zero ("true") result only if both x and y are non-zero.
- we don't know what that non-zero value will be, though.
Otherwise, it produces zero.

The same holds for || and !.
- they will produce either Ø or some non-zero value.

Bitwise operators

- these perform logical operations on <u>individual</u> <u>bits</u> of the operands

  & : bitwise AND

  | : bitwise OR

  ~ : complement (bitwise NOT)

  ^ : XOR (exclusive OR)

Example: For 1-bit numbers

$\underline{\underline{\&}}$

$$\begin{array}{r}0\\0\\\hline0\end{array}\quad\begin{array}{r}1\\0\\\hline0\end{array}\quad\begin{array}{r}0\\1\\\hline0\end{array}\quad\begin{array}{r}1\\1\\\hline1\end{array}$$

$\underline{\underline{|}}$

$$\begin{array}{r}0\\0\\\hline0\end{array}\quad\begin{array}{r}1\\0\\\hline1\end{array}\quad\begin{array}{r}0\\1\\\hline1\end{array}\quad\begin{array}{r}1\\1\\\hline1\end{array}$$

$\underline{\underline{\sim}}$ $\quad \sim 0 = 1 \qquad \sim 1 = 0$

$\underline{\underline{\wedge}}$

XOR

$$\wedge\ \begin{array}{r}0\\0\\\hline0\end{array}\quad\begin{array}{r}1\\0\\\hline1\end{array}\quad\begin{array}{r}0\\1\\\hline1\end{array}\quad\begin{array}{r}1\\1\\\hline0\end{array}$$

4-bit numbers

$$\begin{array}{r} 0101 \\ \& \ 1100 \\ \hline 0100 \end{array} \qquad \begin{array}{r} 0101 \\ | \ \ 1100 \\ \hline 1101 \end{array}$$

You <u>can't</u> use the logical operators &&, ||, etc. and the bitwise operators, &, |, etc. interchangeably.

For example:

x: 1010 "true"

y: 0101 "true"

x && y → non-zero ("true")

x & y → 0000 ("false")

if (x && y)

--- ← will be executed

if (x & y)

--- ← won't be executed.