

Bitwise Operations

CS 2130: Computer Systems and Organization 1

Xinyao Yi Ph.D.
Assistant Professor

Announcement

- Homework 1 due February 2, 2026

Two's Complement

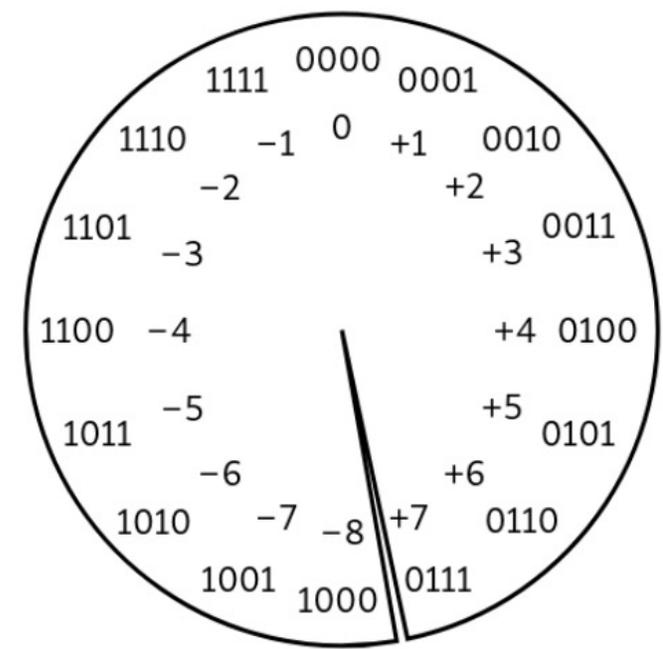
The scheme is called Two's Complement

Why do we need Two's Complement?

- We want the computer to represent both positive and negative numbers.
- And we want addition and subtraction to use the *same* hardware (just one adder), instead of building a separate “subtractor.”

How does it work?

- The **leftmost bit (MSB)** is treated as negative.
 - In normal binary: the leftmost bit is +128 (for 8-bit).
 - In two's complement: the leftmost bit is -128.
- That's why $10000000_2 = -128$ instead of +128.



Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

What is its value in decimal?

Values of Two's Complement Numbers

Consider the following 8-bit binary number in Two's Complement:

11010011

What is its value in decimal?

1. Flip all bits
2. Add 1

Values of Two's Complement Numbers

Why “invert the bits and add 1”?

- Because in 8 bits, we have 256 total values (0–255).
- A negative number is stored as $256 - (\text{its absolute value})$.
- The “invert + 1” trick is just a fast way to compute that.

Values of Two's Complement Numbers

Consider the following decimal number:

-117

What is its value in 8-bit binary binary?

Operations

So far, we have discussed:

- Addition: $x + y$
 - Can get multiplication
- Subtraction: $x - y$
 - Can get division, but more difficult
- Unary minus (negative): $-x$
 - Flip the bits and add 1

Operations (on Integers)

Bit vector: fixed-length sequence of bits (ex: bits in an integer)

- Manipulated by bitwise operations

Bitwise operations: operate over the bits in a bit vector

- Bitwise not: $\sim x$ - flips all bits (unary)
- Bitwise and: $x \& y$ - set bit to 1 if x, y have 1 in same bit
- Bitwise or: $x | y$ - set bit to 1 if either x or y have 1
- Bitwise xor: $x \wedge y$ - set bit to 1 if x, y bit differs

Example: Bitwise AND

```
  11001010  
& 01111100  
-----
```

Example: Bitwise OR

```
    11001010
  | 01111100
            
```

Example: Bitwise XOR

```
  11001010
^ 01111100
-----
```

Your Turn!

What is:

$0x1a \wedge 0x72$

Operations (on Integers)

Logical not: $!x$

- $!0 = 1$ and $!x = 0, \forall x \neq 0$
- Useful in C, no booleans
- Some languages name this one differently

Operations (on Integers)

Left shift: $x \ll y$ - move bits to the left

- Effectively multiply by powers of 2

Right shift: $x \gg y$ - move bits to the right

- Effectively divide by powers of 2
- Signed (extend sign bit) vs unsigned (extend 0)

Left Bit-shift Example

01011010 \ll 2

Right Bit-shift Example

01011010 >> 3

Bit-shift

Computing bit-shift effectively multiplies/divides by powers of 2

Consider decimal:

$$2130 \ll_{10} 2 = 213000 = 2130 \times 100$$

$$2130 \gg_{10} 1 = 213 = 2130 / 10$$

Right Bit-shift Example 2

11001010 >> 1

Right Bit-shift Example 2

For signed integers, extend the sign bit (1)

- Keeps negative value (if applicable)
- Approximates divide by powers of 2

$11001010 \gg 1$

Bit fiddling example

Any Questions?