

Bitwise Operations



CS 2130: Computer Systems and Organization 1
January 23, 2026

Announcements

- Homework 1 due February 2, 2026

Finally, Numbers!

Storing Integers

- Use binary representation of decimal numbers
- Usually have a limited number of bits (ex: 32, 64)
 - Depending on language
 - Depending on hardware

Using Different Bases in Code

	C, assembly Old Languages	Python New Languages
binary	no way	0b1010010110
octal ✗	0732	0o7325
decimal	2130	4256
hexadecimal	0x45af ↑ "hex"	0x4286ab

Java

Finally, Numbers!

Storing Integers

- Use binary representation of decimal numbers
- Usually have a limited number of bits (ex: 32, 64)
 - Depending on language
 - Depending on hardware
- Is there something missing?

Negative Integers

Representing negative integers

Negative Integers

Representing negative integers

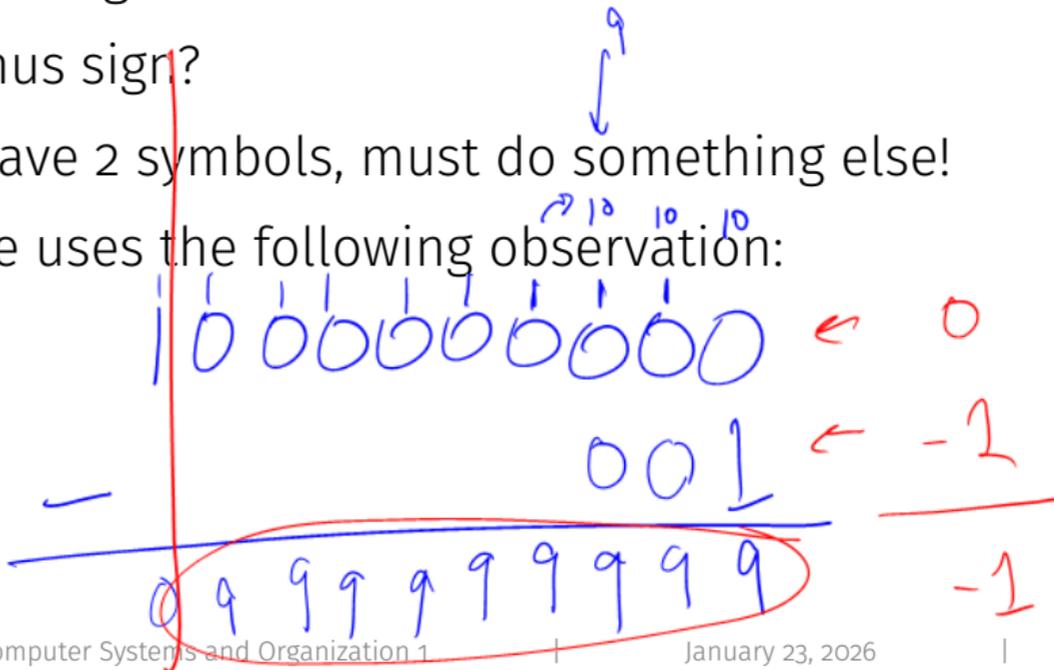
- Can we use the minus sign?

-1010110

Negative Integers

Representing negative integers

- Can we use the minus sign?
- In binary we only have 2 symbols, must do something else!
- Almost all hardware uses the following observation:



Representing Negative Integers

Computers store numbers in fixed number of wires

- Ex: consider 4-digit decimal numbers

$$\begin{array}{r} \overset{10}{1} \overset{10}{1} \overset{10}{1} \overset{10}{1} \\ \overset{1}{1} \overset{1}{1} \overset{1}{1} \overset{1}{1} \\ 0000 \\ - 0001 \\ \hline 9999 \leftarrow -1 \\ - 1 \\ \hline 9998 \leftarrow -2 \end{array}$$

Representing Negative Integers

Computers store numbers in fixed number of wires

- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
 - $0000 - 0001 = 9999 == -1$
 - $9999 - 0001 = 9998 == -2$
 - Normal subtraction/addition still works
 - Ex: $-2 + 3$

Handwritten arithmetic showing the addition of 9999 and 3, resulting in 0001. The carry outs are indicated by three '1's above the 9998 and a '3' below the 8. A red arrow points from the text 'carry out' to the first '1'.

$$\begin{array}{r} 111 \\ 9998 \\ + \quad 3 \\ \hline 0001 \end{array}$$

Representing Negative Integers

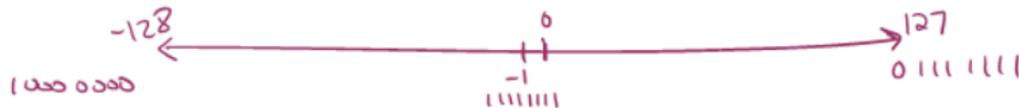
Computers store numbers in fixed number of wires

- Ex: consider 4-digit decimal numbers
- Throw away the last borrow:
 - $0000 - 0001 = 9999 \equiv -1$
 - $9999 - 0001 = 9998 \equiv -2$
 - Normal subtraction/addition still works
 - Ex: $-2 + 3$
- This works the same in binary

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad 1 \\ \quad 0 \quad 0 \quad 0 \quad 0_2 \\ - \quad 0 \quad 0 \quad 0 \quad 1 \\ \hline 1 \quad 1 \quad 1 \quad 1 \end{array} = 0_{10} - 1_{10} = -1_{10}$$

Two's Complement

Values of Two's Complement Numbers



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

↓
11010011

What is its value in decimal?

- how far is it from 0?
- how far is it from -1?

$$\begin{array}{r} 1111\ 1111 \\ - 1101\ 0011 \\ \hline 0010\ 1100 \\ + 1 \\ \hline 0010\ 1101 = 45 \end{array}$$

32 16 8 4 2 1

$$\begin{array}{r} -1 \\ - 245 \\ \hline 44 \end{array}$$

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

Addition

$$\begin{array}{r} 1001010 \\ + 1111100 \\ \hline 1000110 \\ \text{overflow} \end{array}$$

Subtraction

$$\begin{array}{r} 1001010 \\ - 0111100 \\ \hline 1100010 \end{array}$$

Handwritten annotations in blue ink:

- Vertical lines above the minuend (01001010) at the 2nd, 3rd, 4th, 5th, and 6th positions from the right, with the label "10" written above each line.
- A vertical line above the minuend at the 7th position from the right, with the label "10" written above it.
- A vertical line above the minuend at the 8th position from the right, with the label "1" written above it.

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

$$\begin{array}{r} 11001010 \\ \& 01111100 \\ \hline 01001000 \end{array}$$

Example: Bitwise OR

```
  11001010  
| 01111100  


---


```

Example: Bitwise XOR

$$\begin{array}{r} 11001010 \\ \wedge 01111100 \\ \hline \end{array}$$

Your Turn!

What is: $0x1a \hat{\ } 0x72$