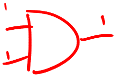
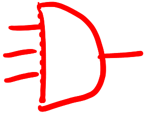
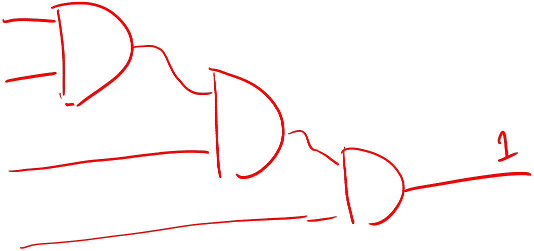
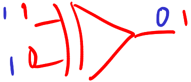
Computer Systems and Organization 1



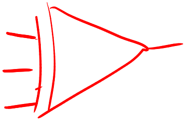
Warm up!
Can I make an n-input AND from 2-input AND gates?

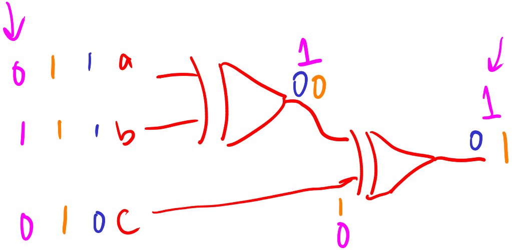






Warm up!
What about XOR gates?





Parity - 0 if even # of bits = 1
1 if odd # bits = 1

More bits, circuits, adders

CS 2130: Computer Systems and Organization 1

September 10, 2025

Announcements

- Homework 1 due Monday

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: `~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 ^ y` - set bit to 1 if x, y bit differs

Operations (on Integers)

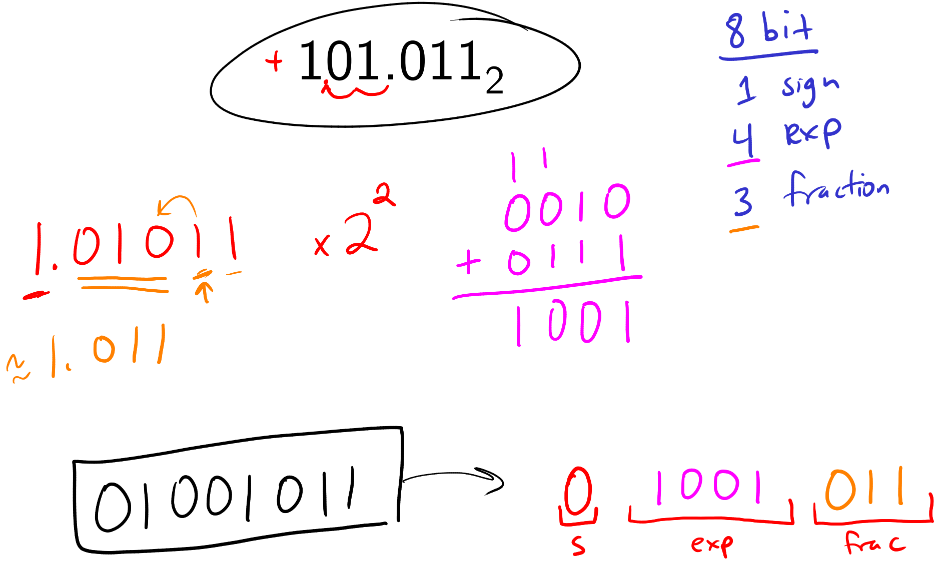
- Logical not: `!x`
  - $!0 = 1$ and $!x = 0, \forall x \neq 0$
  - Useful in C, no booleans
  - Some languages name this one differently
- Left shift: `x << y` - move bits to the left
  - Effectively multiply by powers of 2
- Right shift: `x >> y` - move bits to the right
  - Effectively divide by powers of 2
  - Signed (extend sign bit) vs unsigned (extend 0)

Floating Point Example

$$101.011_2$$

Floating Point Example

101.011_2



Floating Point Example

What does the following encode?

1	001110	1010101

What about 0?

Floating Point Numbers

Four cases:

- **Normalized**: What we have seen today

$$s\ eeee\ ffff = \pm 1.ffff \times 2^{eeee - \text{bias}}$$

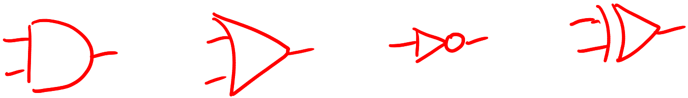
- **Denormalized**: Exponent bits all 0

$$s\ eeee\ ffff = \pm 0.ffff \times 2^{1 - \text{bias}}$$

- **Infinity**: Exponent bits all 1, fraction bits all 0 (i.e., $\pm\infty$)
- **Not a Number (NaN)**: Exponent bits all 1, fraction bits not all 0

Our story so far

- Transistors
- Information modeled by voltage through wires (1 vs 0)
- Gates: & | ~ ^



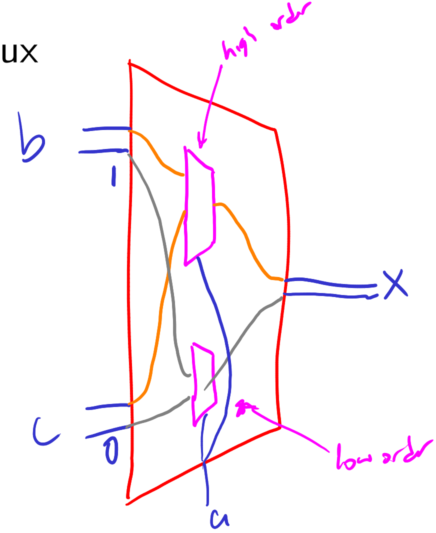
- Multi-bit values: representing integers
  - Signed and unsigned
  - Bitwise operators on bit vectors
- Floating point

How to do the *work* of multi-bit?

Multi-bit Mux

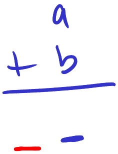
Our first multi-bit example: mux

$x = a\ ?\ b : c$

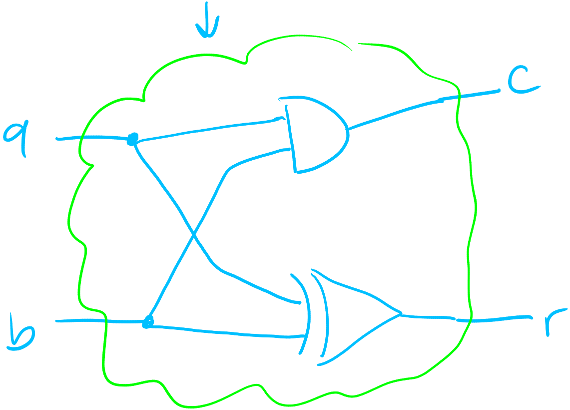


Adder

Add 2 1-bit numbers: a, b



a	b	c	r
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



Adder

Can we use this in parallel to add multi-bit numbers?

Adder

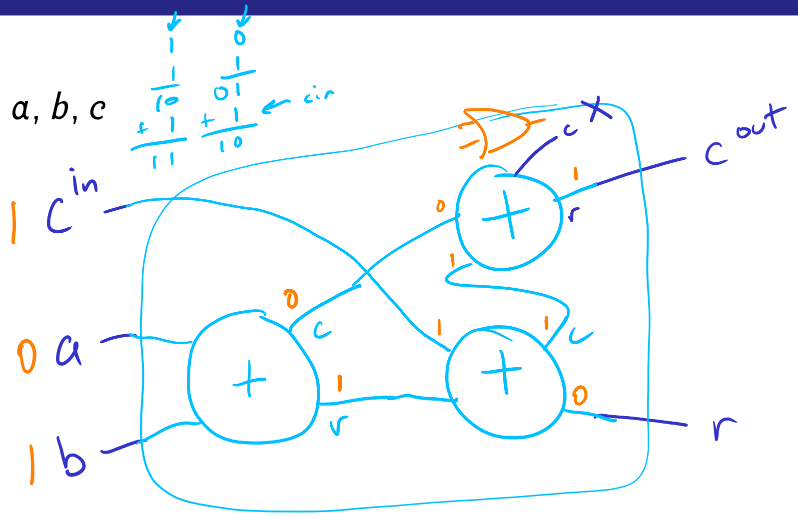
Can we use this in parallel to add multi-bit numbers? What is missing?
Consider:

$$\begin{array}{r} 11 \\ +01 \\ \hline \end{array}$$

3-input Adder

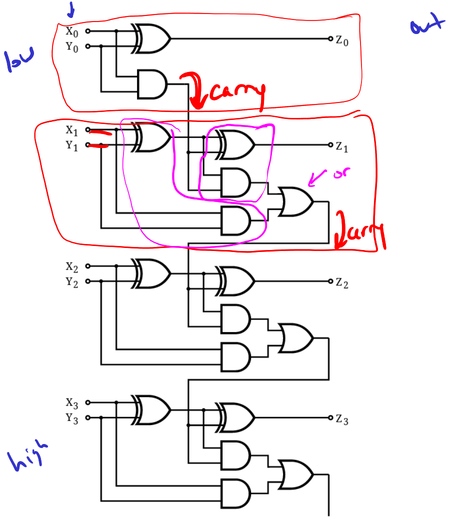
Add 3 1-bit numbers: a, b, c

a	b	c^{in}	c^{out}	r
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

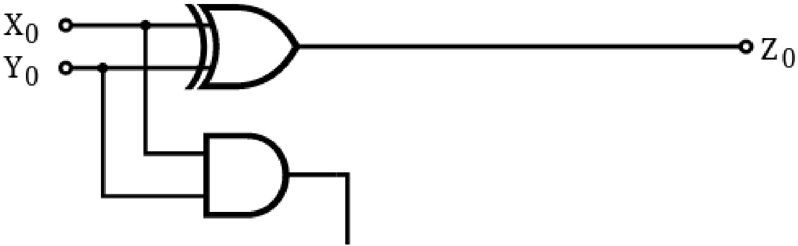


Ripple-Carry Adder

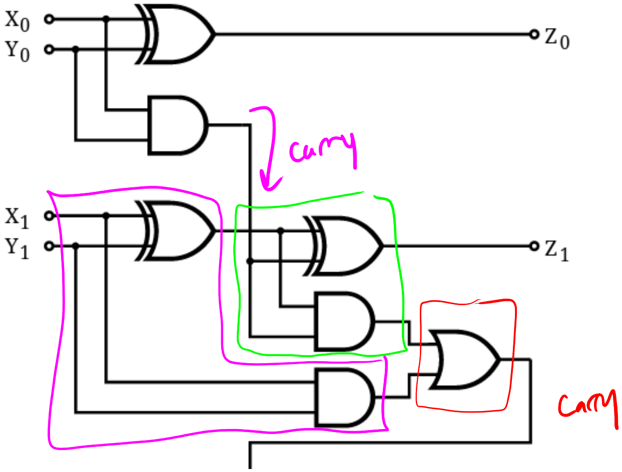
$$\begin{array}{r} X_3 X_2 X_1 X_0 \\ +\ Y_3 Y_2 Y_1 Y_0 \\ \hline Z_3 Z_2 Z_1 Z_0 \end{array}$$



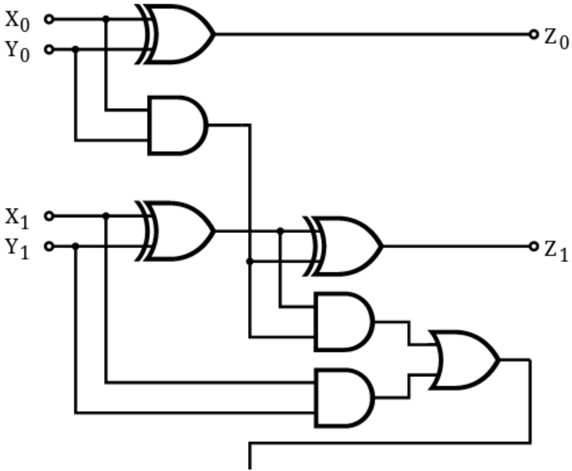
Ripple-Carry Adder: Lowest-order Bit



Ripple-Carry Adder: In General



Ripple-Carry Adder: In General



What does this circuit do?