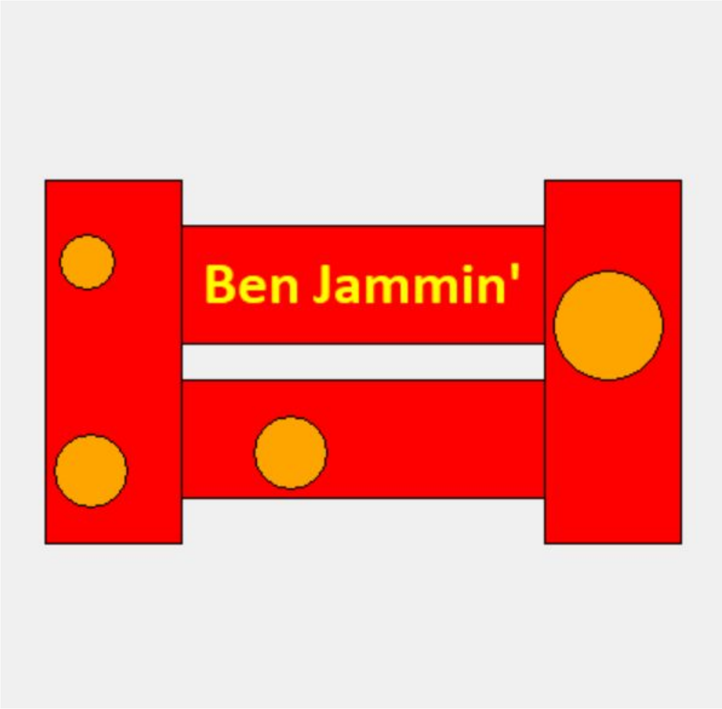
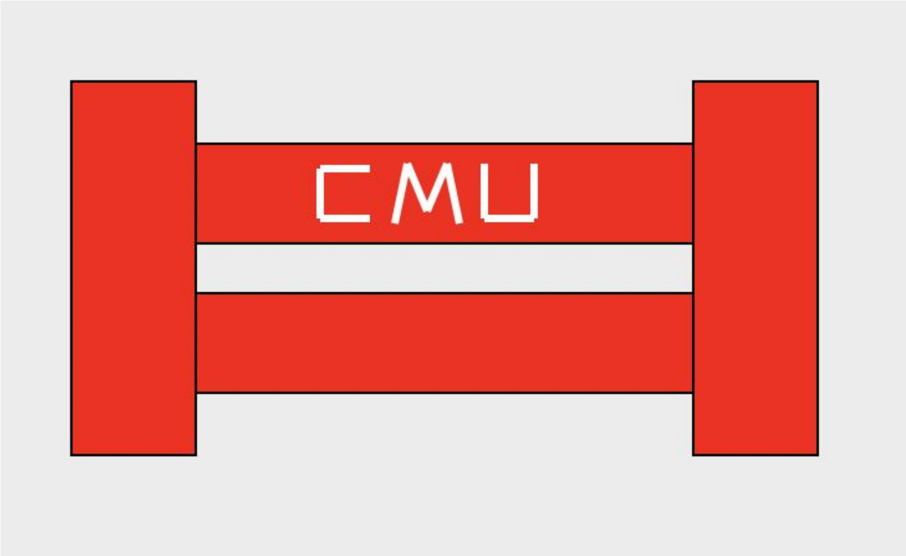
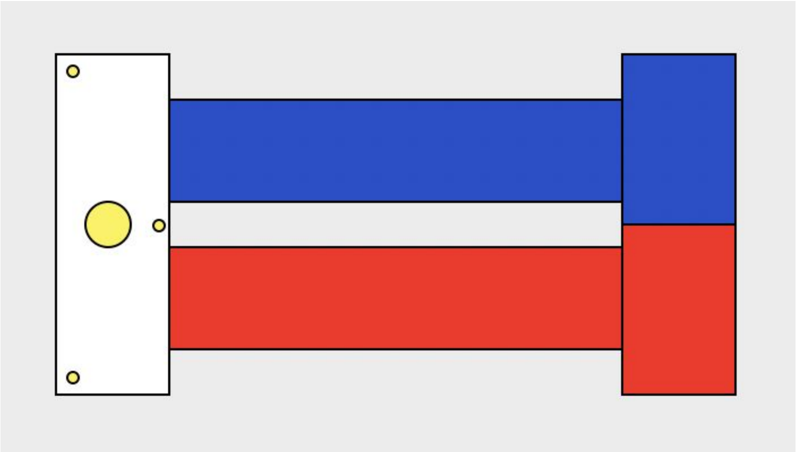
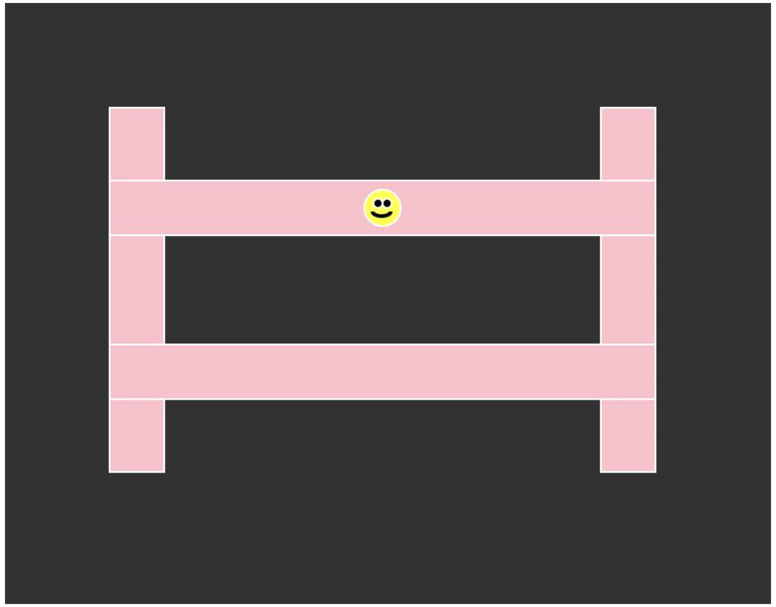
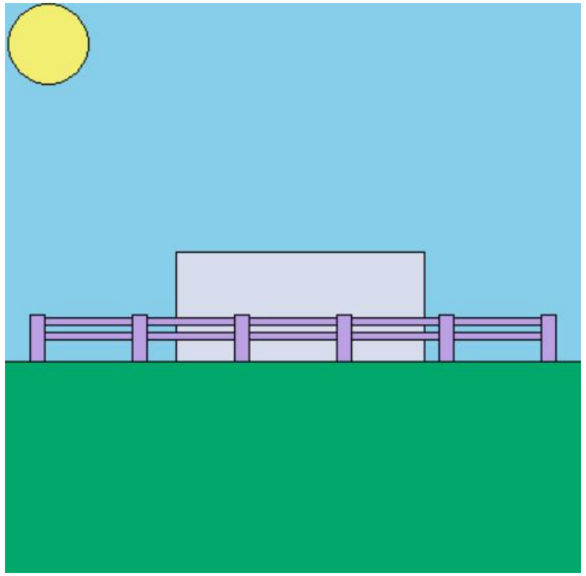
Hw1 – Awesome Fences!

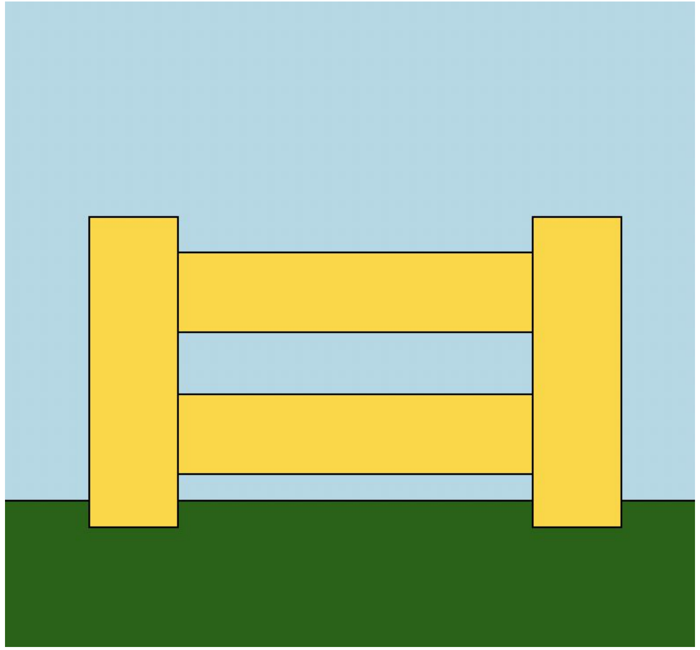


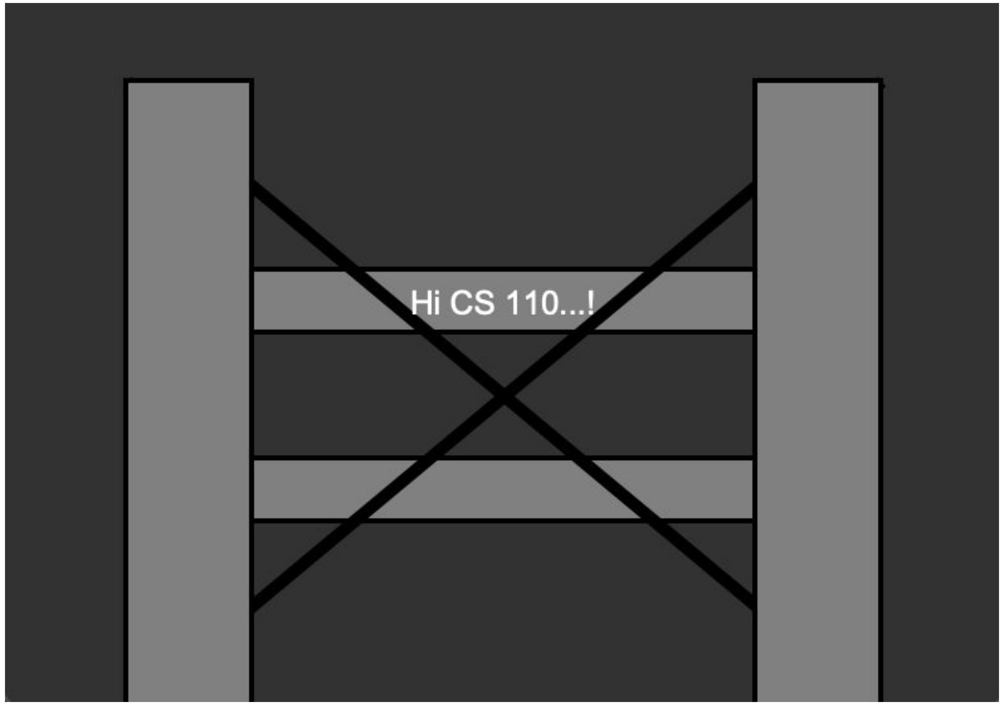


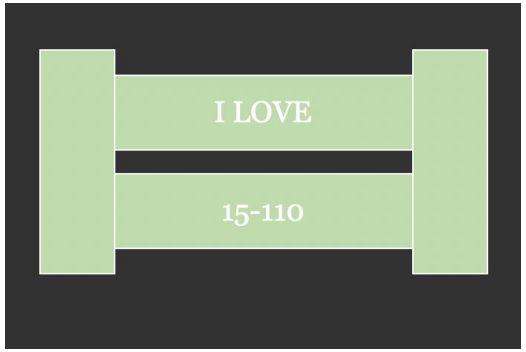


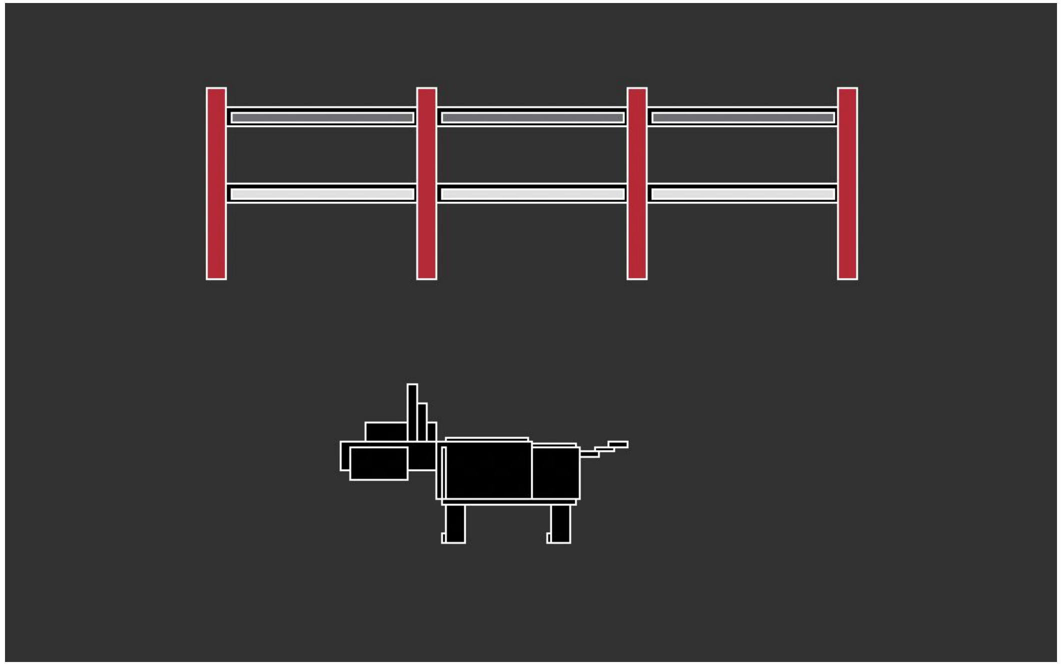


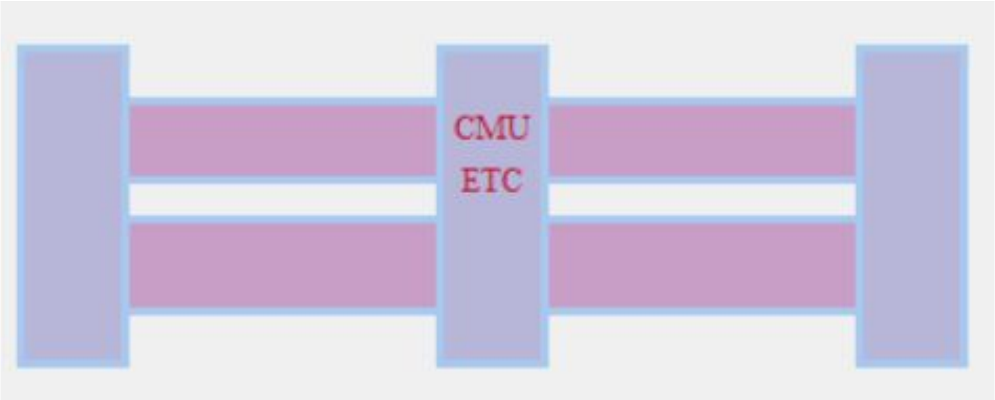





Hi CS 110...!


I LOVE
15-110




CMU
ETC

Announcements

- Hw1 feedback released
  - Make sure to view programming feedback!
  - Tutorial on website
- For check2 and beyond, look at your autograder feedback!
  - Start early, work through sections right after we cover new material

Quizlet

Circuits and Gates

15-110 – Wednesday 09/11

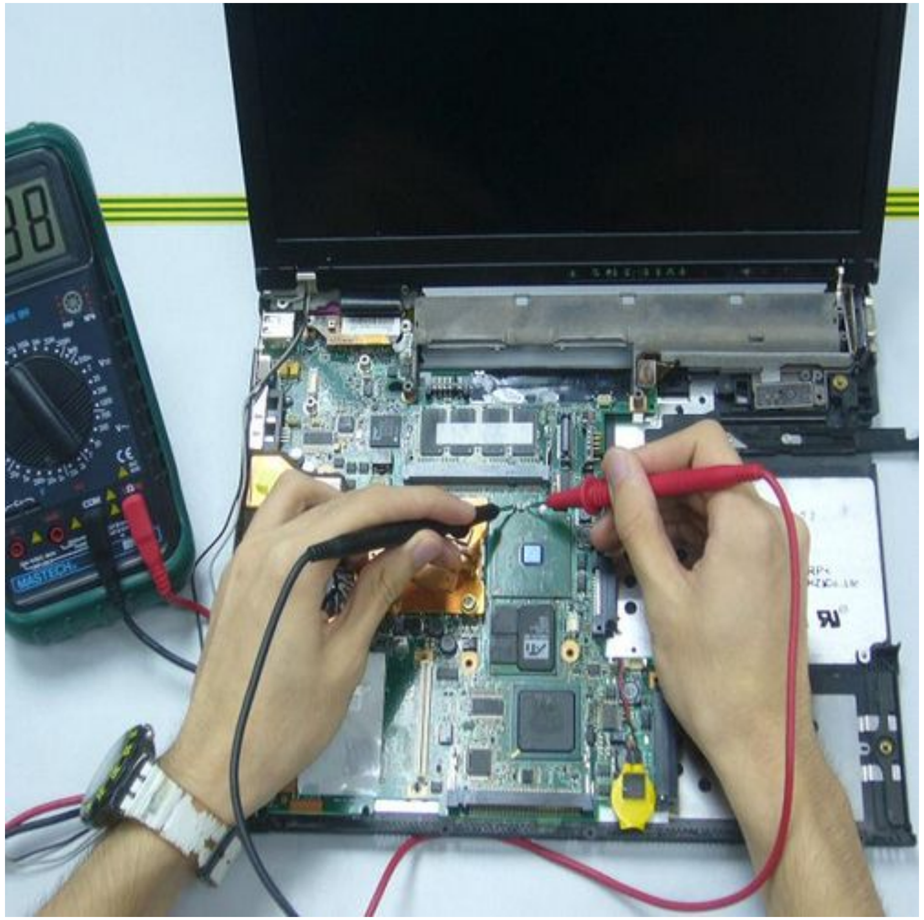
Learning Goals

- Translate **Boolean expressions** to **truth tables** and **circuits**
- Translate **circuits** to **truth tables** and **Boolean expressions**
- Recognize how addition is done at the circuit level using algorithms and abstraction

All the operations we perform on a computer correspond to physical actions within the **hardware** of the machine.

Software: the abstracted concepts of computation- how computers represent data, and how programs can manipulate data.

Hardware: the actual physical components used to implement software, like the laptop components shown to the right.

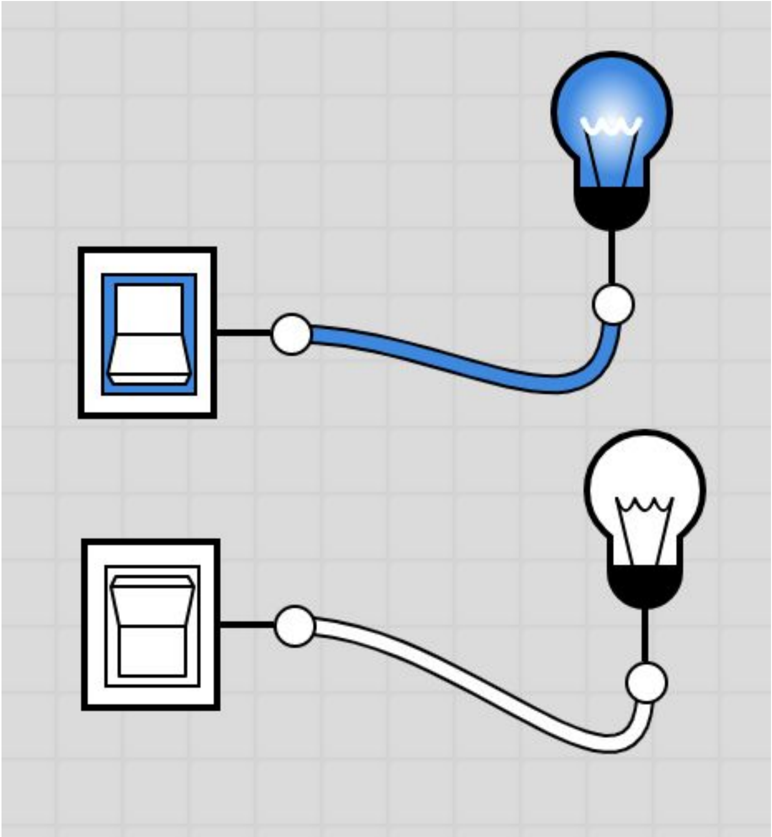


In hardware, bits are represented as **electrical voltage**.

A high level of voltage is considered a **1**.

A low level of voltage is considered a **0**.

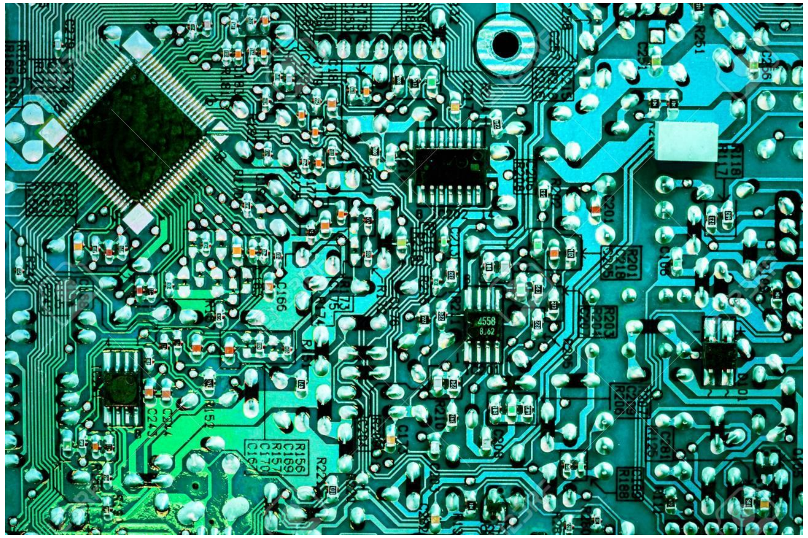
By redirecting electrical flow throughout a system, we can change the values of data in hardware.

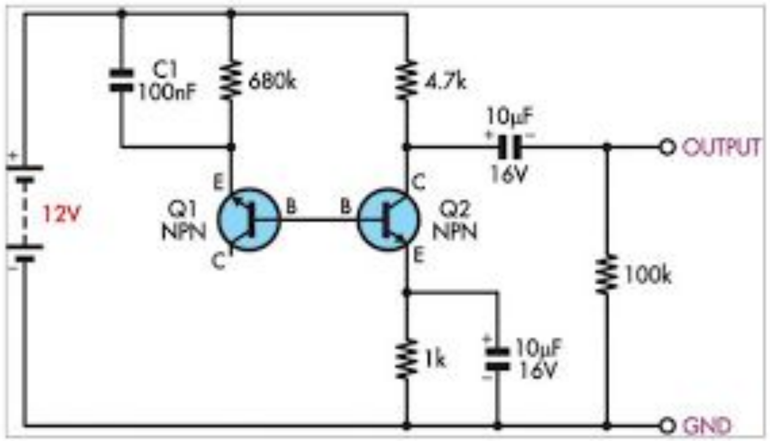


The computer uses **circuits** to perform computational actions.

Circuits redirect electricity to different parts of hardware.

We will discuss how to use **gates**, which are abstracted circuit components. Every gate we discuss can be directly translated to a real hardware circuit.





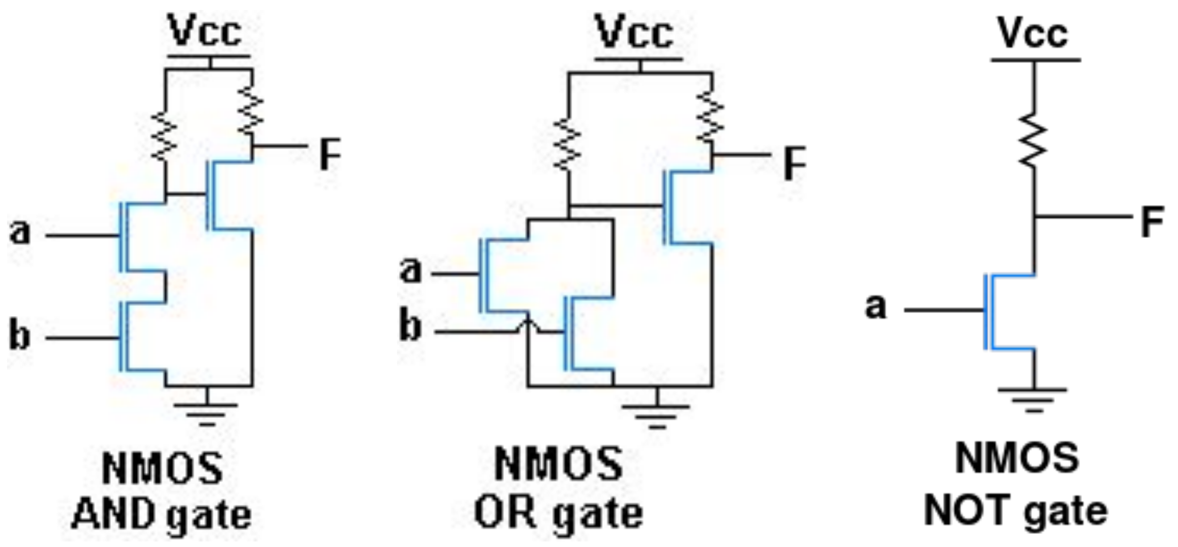
Logical Gates

Our **three basic logical operators** (`and`, `or`, `not`) can be represented as gates in **real hardware**. 1 is `True` and 0 is `False`.

AND gate takes two inputs and outputs 1 if both inputs are 1

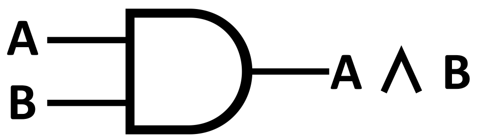
OR gate takes two inputs and outputs 1 if either inputs are 1

NOT gate takes one inputs and outputs the reverse, 1 becomes 0 and 0 because 1



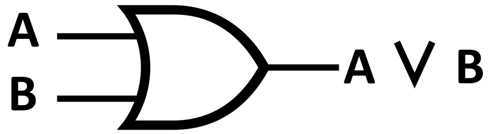
We will use **special symbols** to represent building circuits with these gates:

AND gate takes two inputs and outputs 1 if both inputs are 1



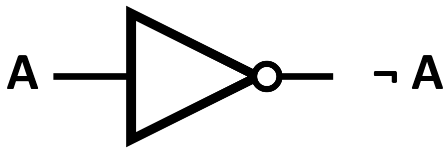
A	B	A ∧ B
1	1	1
1	0	0
0	1	0
0	0	0

OR gate takes two inputs and outputs 1 if either inputs are 1



A	B	A ∨ B
1	1	1
1	0	1
0	1	1
0	0	0

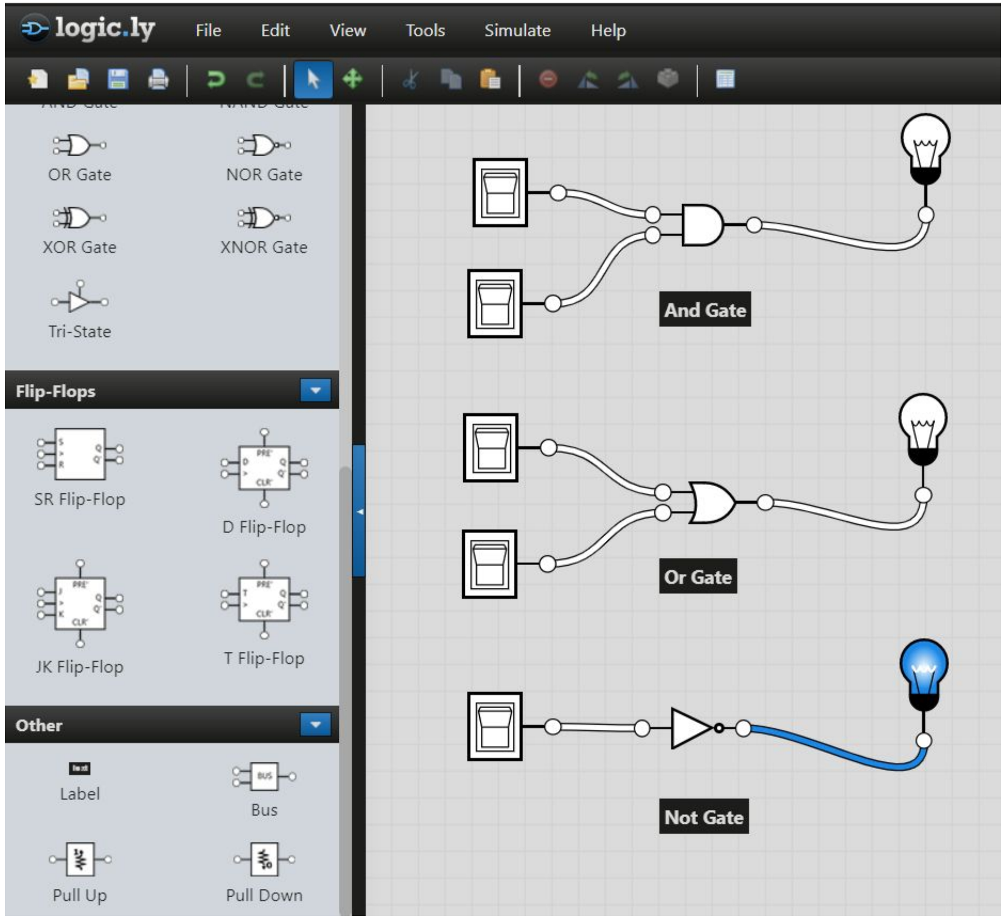
NOT gate takes one inputs and outputs the reverse, 1 becomes 0 and 0 because 1



A	¬ A
1	0
0	1

When working with gates, it can help to **simulate a circuit** using the gates to investigate how they work.

There are lots of free online circuit simulators. We'll use this one: https://logic.ly/demo

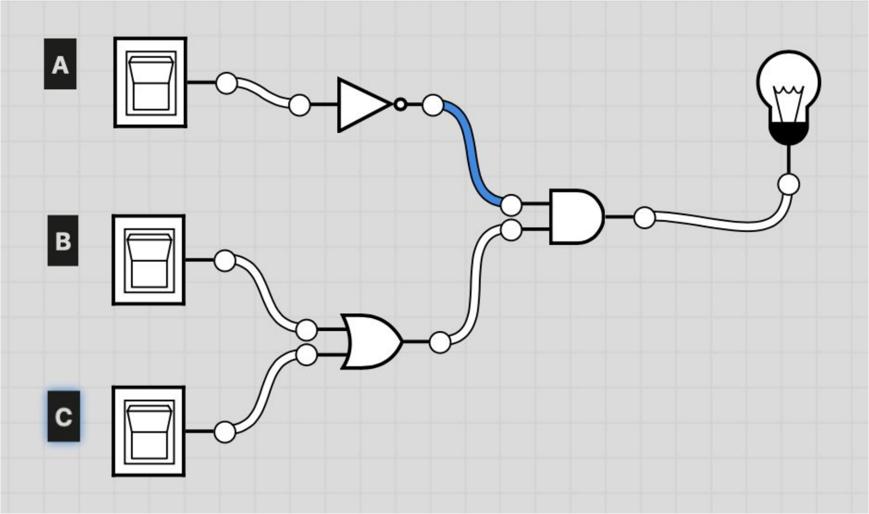


Algorithms with Gates

We can combine gates together in different orders to achieve different results. This lets us **build algorithms using gates**.

When we want to represent an algorithm that uses gates, we can use one of three different representation formats:

Circuit



Truth Table

A	B	C	Output
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

Boolean expression

$\neg A \land (B \lor C)$

We can use truth tables to show **all the possible inputs and outputs** of expressions.

All possibilities for the expression **A ∨ ¬B**

Inputs		Outputs	
A	**B**	**¬B**	**A ∨ ¬B**
1	1	0	1
1	0	1	1
0	1	0	0
0	0	1	1

As a Boolean expression: `A or (not B)`

With truth tables we can **break down complex expressions** into smaller parts and give each part its own column.

$$(A \land B \land C) \lor (A \land \neg B \land \neg C) \lor (\neg A \land B \land \neg C) \lor (\neg A \land \neg B \land C)$$

Inputs			Outputs				
A	B	C	A ∧ B ∧ C	A ∧ ¬B ∧ ¬C	¬A ∧ B ∧ ¬C	¬A ∧ ¬B ∧ C	(A ∧ B ∧ C) ∨ (A ∧ ¬B ∧ ¬C) ∨ (¬A ∧ B ∧ ¬C) ∨ (¬A ∧ ¬B ∧ C)
1	1	1	1	0	0	0	1
1	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	0	0	0	1	0	0	1
0	1	1	0	0	0	0	0
0	1	0	0	0	1	0	1
0	0	1	0	0	0	1	1
0	0	0	0	0	0	0	0

Boolean Expressions, Circuits, and Truth Tables can all be used to represent the **same algorithm**.

Boolean Expressions:

Good for quickly representing an algorithm in text

Circuits:

A more visual option, and more interactive

Truth Tables:

Lay out all inputs and outputs, which helps derive algorithms

Truth table -> Boolean expressions

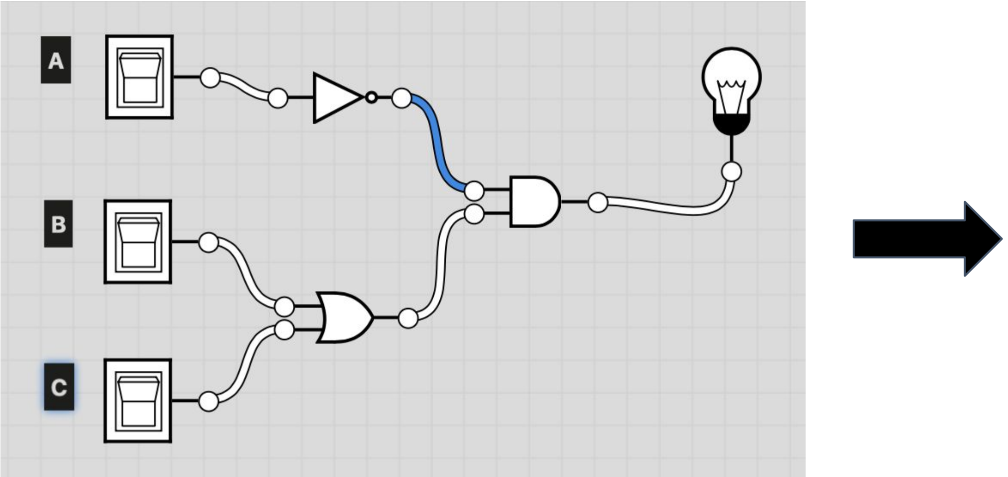
We can use a truth table to **derive** a Boolean expression from a set of inputs and outputs.

This is too complex for this class!

A	B	C	??
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

Circuit -> Truth table

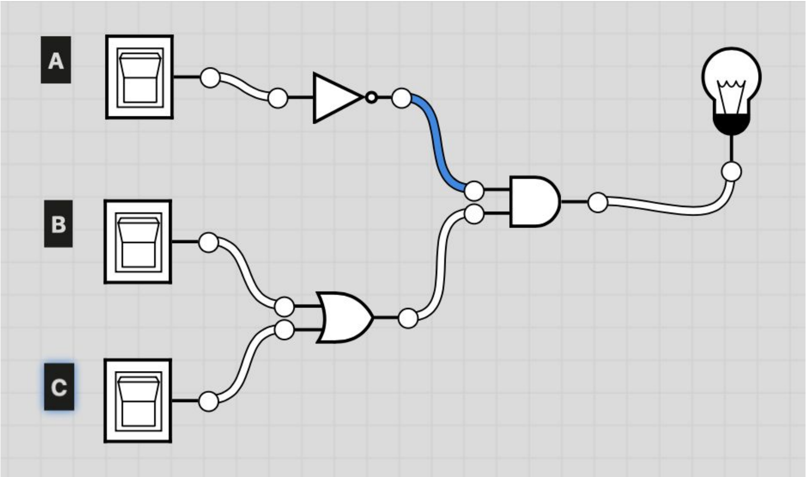
Given a circuit, we can construct a truth table either by logically determining the result, or by simulating all possible input combinations.



A	B	C	Output
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

Circuits -> Boolean expression

We can find the equivalent Boolean expression by translating gates to Boolean operators.

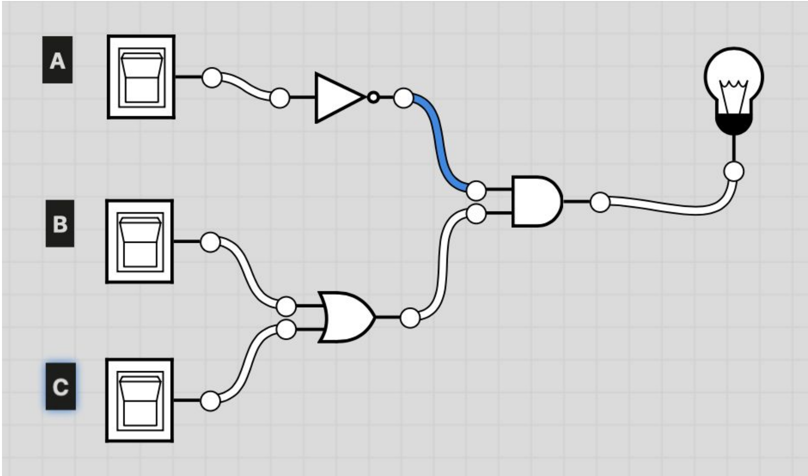


$\neg A \land (B \lor C)$

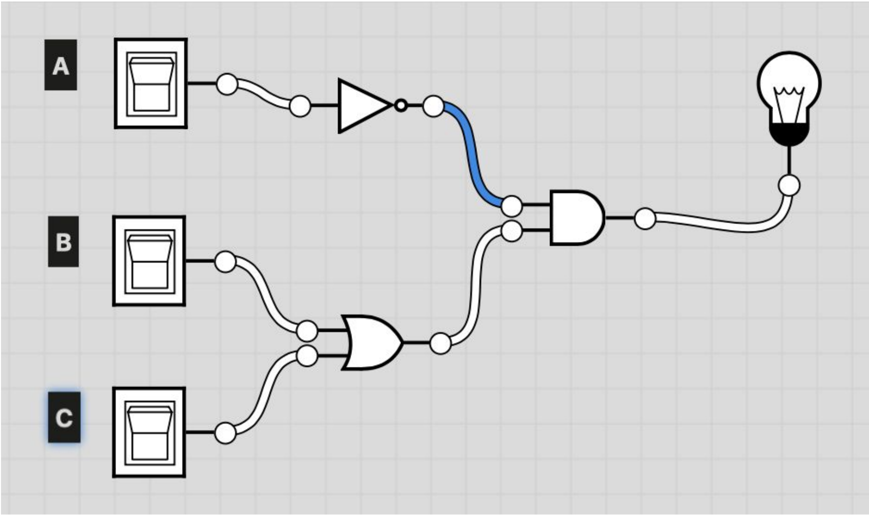
Boolean expression -> circuit

We can also go in the opposite direction.

$\neg A \land (B \lor C)$



Conversion Chart



Try all possible input combinations

Use problem solving and logic

A	B	C	Output
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	0

Convert gates to Boolean operators

Convert Boolean operators to gates

Try all possible input combinations

Use problem solving and logic

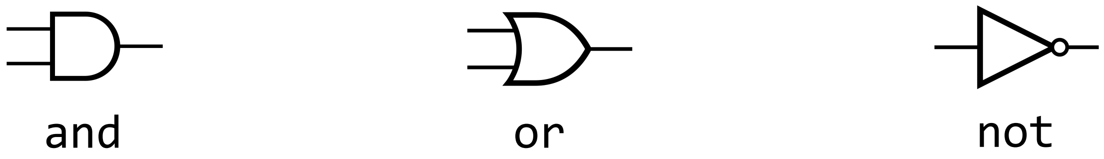
$\neg A \land (B \lor C)$

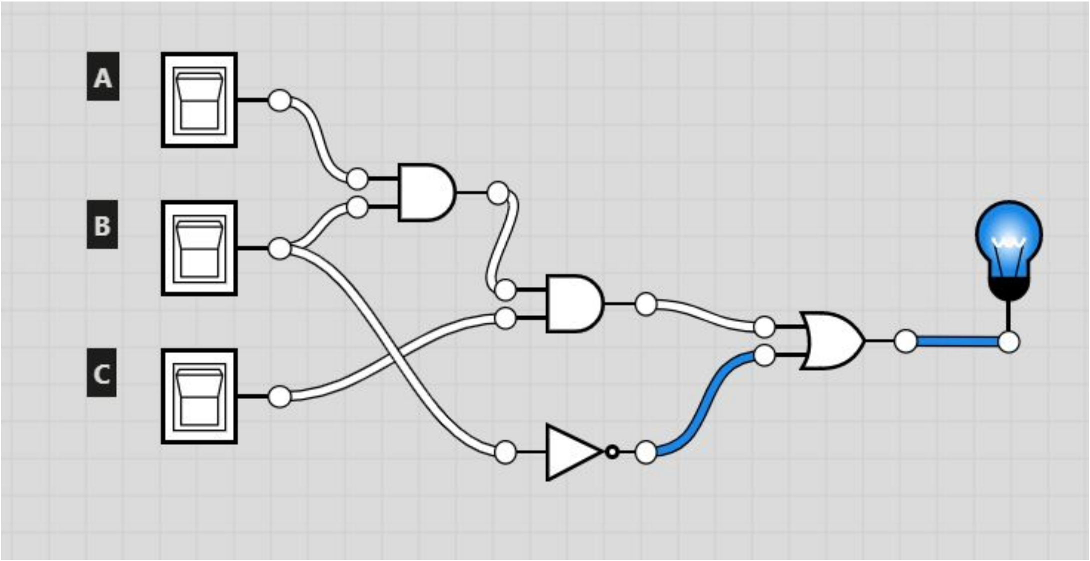
In this class, we **will not require** you to do conversions that require complex problem solving and logic!

Activity: Find the positive inputs!

Convert the following circuit to the equivalent Boolean Expression, then write the equivalent truth table.

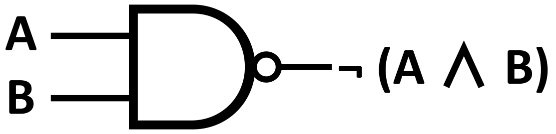
Which input combinations will result in the circuit outputting 1 (the light bulb lighting up)?





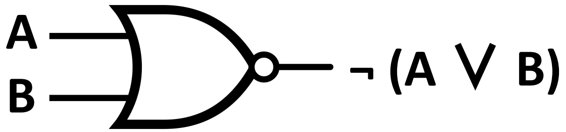
There are **more gates** that can simplify complex circuits:

NAND gate takes two inputs and outputs 1 if both inputs are **not** 1



A	B	¬ (A ∧ B)
1	1	0
1	0	1
0	1	1
0	0	1

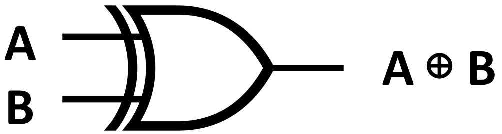
NOR gate takes two inputs and outputs 1 if both inputs are 0



A	B	¬ (A ∨ B)
1	1	0
1	0	0
0	1	0
0	0	1

XOR gate takes two inputs and outputs 1 if one input 1 and the other is 0:

$$(A \land \neg B) \lor (\neg A \land B)$$



A	B	A ⊕ B
1	1	0
1	0	1
0	1	1
0	0	0

Abstraction with Gates

We can write real **algorithms with circuits**.

We'll focus on a basic action that computers do all the time: **integer addition.**

How do we add two one-bit numbers, X and Y? What are all the possible inputs and outputs?

In binary:

1 + 1 = 10

X	Y	X + Y
1	1	10
1	0	01
0	1	01
0	0	00

We need **two bits** to store the result!

We implement single-bit addition with gates using a **half adder**.

We need **two bits** to hold the result which we call:

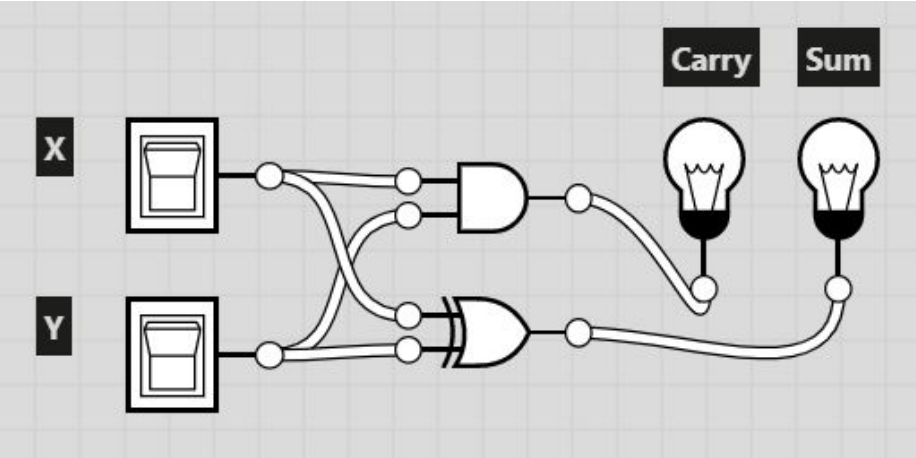
$$1 + 1 = \boxed{1}\ \boxed{0}$$

carry bit **sum bit**

X	Y	X+Y
1	1	10
1	0	01
0	1	01
0	0	00

Carry	Sum
1	0
0	1
0	1
0	0

$X \wedge Y$	$X \oplus Y$
1	0
0	1
0	1
0	0



Sum bit is an **XOR** gate and carry is an **AND** gate.

To handle numbers with multiple bits, we might need to carry an **output over to the next column** of the addition.

For the two's column on the right, call the **carried-in bit C_{in}** and next **carry C_{out}**.

We need to modify our half-adder to have a third input C_{in} and update the computations for Carry (C_{out}) and Sum.

```
            Cout    Cin
  1 1 1 1 1          <- carried bits
    0 1 1 0 1
+   1 0 1 1 1
--------------
=1 0 0 1 0 0 = 36
```

We implement multiple-bit addition with gates using **full adders.**

We now have **3 inputs:**

C_{in}, X, and Y

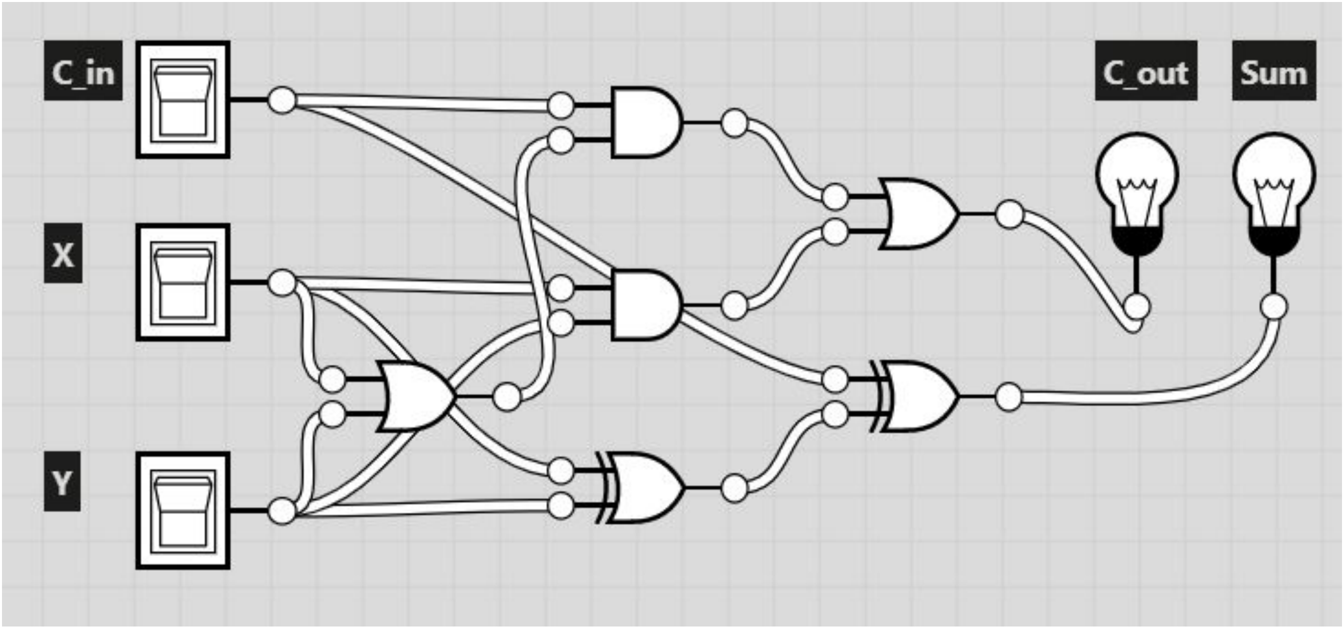
And **2 outputs:**

C_{out} is equivalent to:

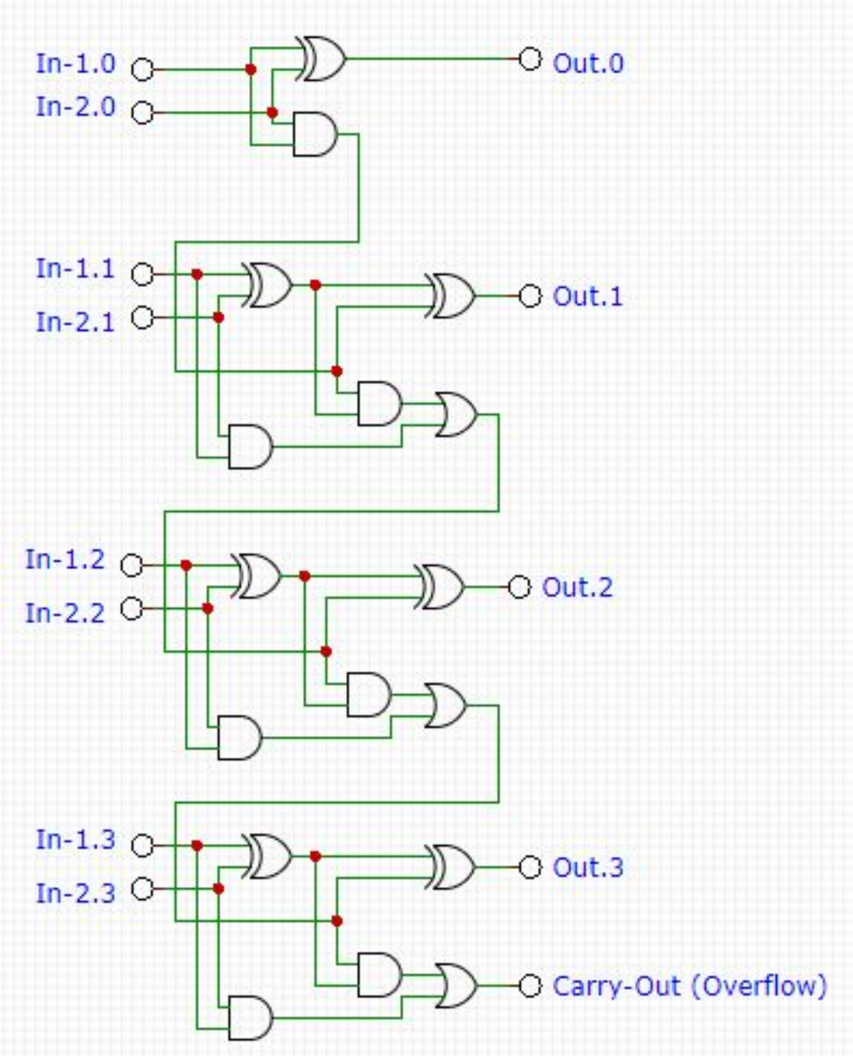
$((X \lor Y) \land C_{in}) \lor (X \land Y)$

Sum is equivalent to:

$(X \oplus Y) \oplus C_{in}$

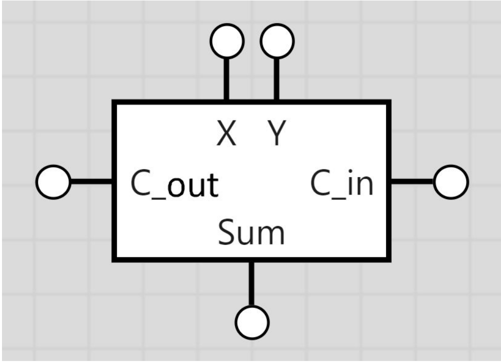


We can implement an N-bit by **chaining together** full adders.



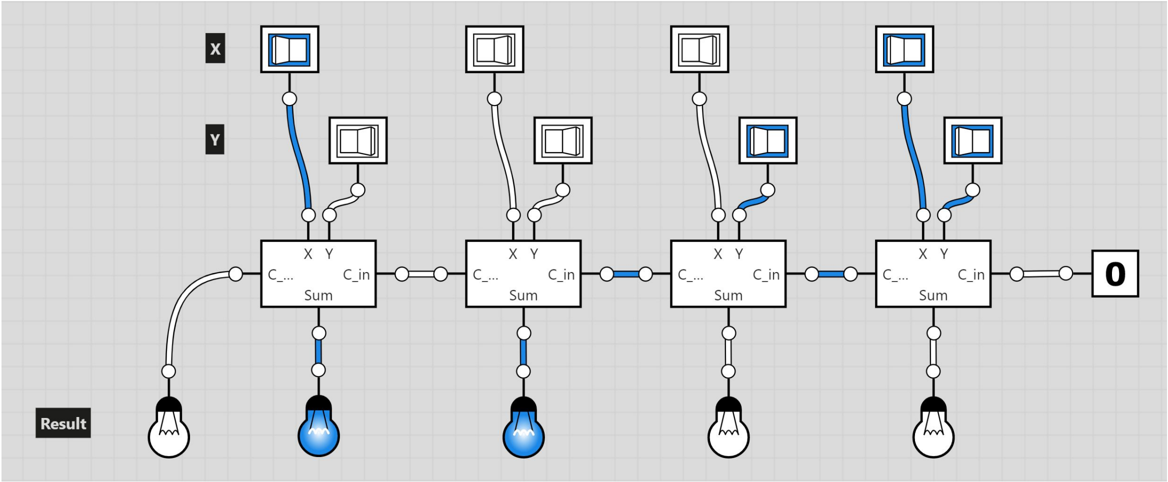
Using **abstraction**, we can represent full adders as a box.

The box holds the Full Adder circuit within it, but it doesn't need to bother with all the internal components.



Let's try it out: What's 9 + 3?

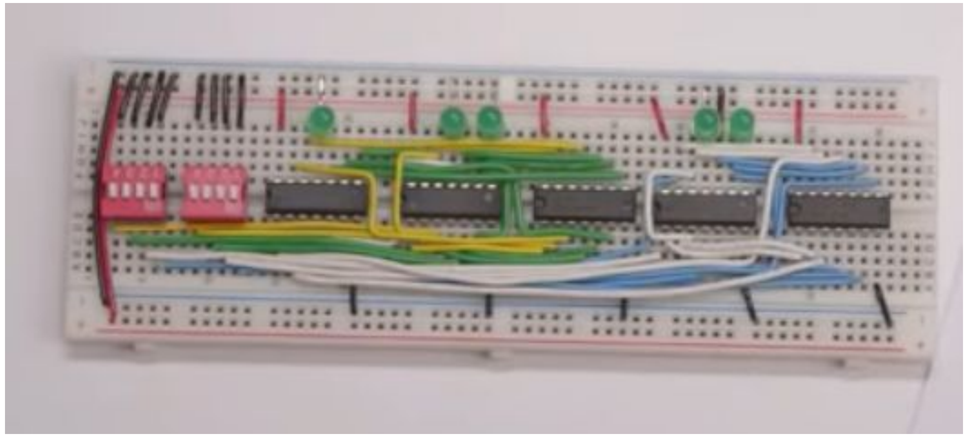
- 9 is 8+1=1001, 3 is 2+1=0011
- Walk through the full adders...
- The output is 1100=8+4
- That's 12! It works!



Sidebar: see a 4-bit Adder in Hardware

You can use the abstract circuit we've designed to build an **actual hardware circuit** that does 4-bit addition (or more!).

See a demo of what that looks like here: https://youtu.be/wvJc9CZcvBc?t=742



Learning Goals

- Translate **Boolean expressions** to **truth tables** and **circuits**
- Translate **circuits** to **truth tables** and **Boolean expressions**
- Recognize how addition is done at the circuit level using algorithms and abstraction