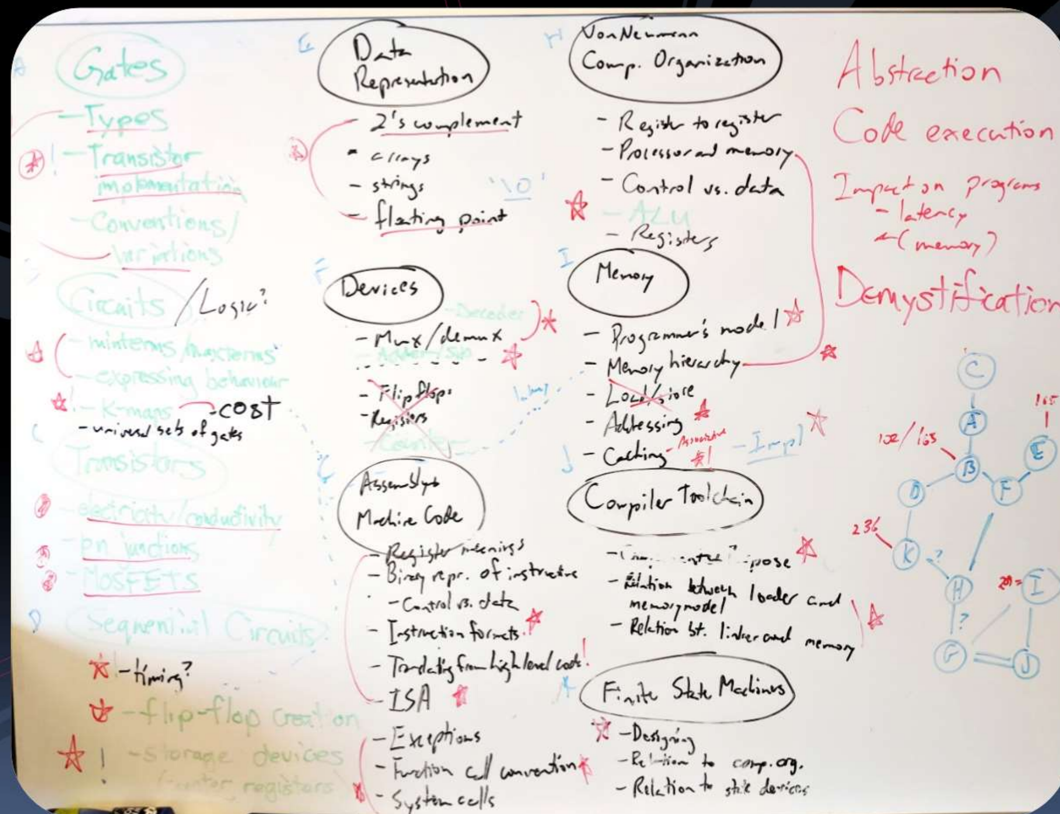
CSC258: Computer Organization

Instructors: Steve Engels, sengels@cs.toronto.edu
Joe Lim, joe.lim@toronto.edu

Breaking down CSC258



CSC258 Course Details

- Lectures
  - Lectures cover topics (generally one per week)
  - Each week builds on the week before
- Tutorials
  - 30 minutes topic review (from previous week)
  - 30 mintues lab prep (for following week)

CSC258 Course Details

- Labs (28%):
  - 7 total (4% each)
  - Must complete pre-lab exercises ahead of time (1%) and demonstrate completed lab tasks to TAs in the lab rooms (BA3145, BA3155, BA3165).
  - Must work in pairs, in your assigned room.
  - Your partner and your lab station are yours for the duration of all 7 labs.
  - No eating or drinking allowed in the lab rooms.

CSC258 Course Details

- Project (14%):
  - Project proposal (2%)
  - 3 milestone demos (4% each)
  - Goal: Large, cool digital creation.
- Exams:
  - Midterm (18%) – Oct 25^{th}, 7pm-9pm, EX100
    - Email us ASAP if you have conflicts.
  - Final exam (40%)
    - Must get 40% to pass the course.

Finally, two common questions

What is the point of this course?

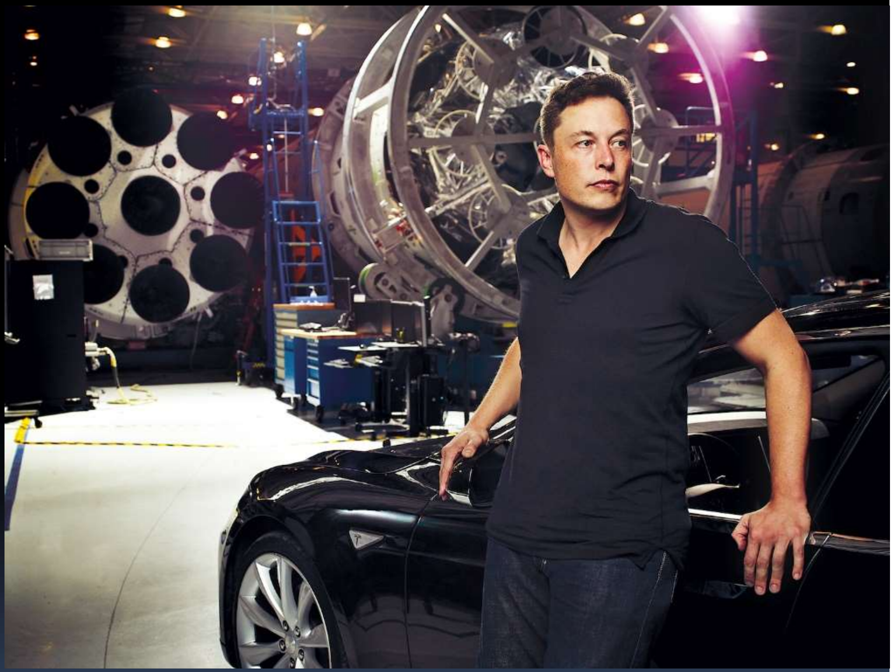
Why are you making me take this?

“Why are you making me take this?”

- CSC258 isn’t needed if you’re just a causal technology user.
  - You can still drive a car, even if you don’t understand how the engine works.

“Why are you making me take this?”

- Computer science majors aren’t casual technology users.
  - At the very least, you’ll need to know how the programs you write are affected by hardware.
  - Processor knowledge is needed for OS courses.
  - Assembly language is needed for low-level tasks like compilers.



“What is the point of this course?”

- Course outcomes:
  - Understand the underlying architecture of computer systems.
  - Learn how to use this architecture to store data and create behaviour.
  - Use the principles of hardware design to create digital logic solutions to given problems.

“What is the point of this course?”

- Our course goals:
  - Make you a better computer scientist.
  - Expose you to new programming paradigms.
  - Give deeper insights into past/current courses.
  - Help prepare you for future courses.
- How we do this:
  - Show you how your computer works.
  - Start from electricity, end with assembly.

Let the learning begin

A few questions to start

- How much do you actually understand about your computer and the programs you run?
- For instance:
  1. When you set a variable to "`false`" or "`true`", how are these boolean values stored?
  2. Why is there a maximum value for an `int`?
  3. Is it cheaper to do an addition operation or a multiplication?
  4. How do boolean operations like "`and`", "`or`" and "`not`" work?
  5. What happens when you press `Ctrl-Alt-Delete`?
  6. What happens when you compile a Java or C program?
  7. What causes a blue screen error on your computer?

CSC258 has the answers

- Computers are physical things, therefore they have certain behaviours and limitations:
  - Data values are finite.
  - All data is stored as ones and zeroes at some level.
  - Many high-level operations depend on low-level ones.

- The way computers are today take their origins from how computers were created in the past.

Example #1: Booleans

- How are boolean values stored?
- Example: `if` statements:

```
if x:
   print 'Hello World'
   # what values can x have
   # that make this happen?
```

  - What if `x` is a boolean?
  - What if `x` is an int?
  - What if `x` is a string?

All comes down to hardware in the end!

Example #2: Integers

- How are `int` values stored?
  - Again, as ones and zeroes.

Decimal → `1234 → 0011000000111001` ← Binary

- How many values can integers have?
  - This can vary based on language and architecture, but generally integers have 2^{32} different values.
  - <u>Signed integers</u>: range from -2^{31} to $2^{31}-1$
  - <u>Unsigned integers</u>: range from 0 to $2^{32}-1$
  - Different ranges for `long`, `short` and `byte`.

What does it all mean?

- Computers do on the hardware level what programs do on the software level.

- It's important to understand that software is like the outer layer to the underlying hardware, which can affect its behaviour.

Programming parallels

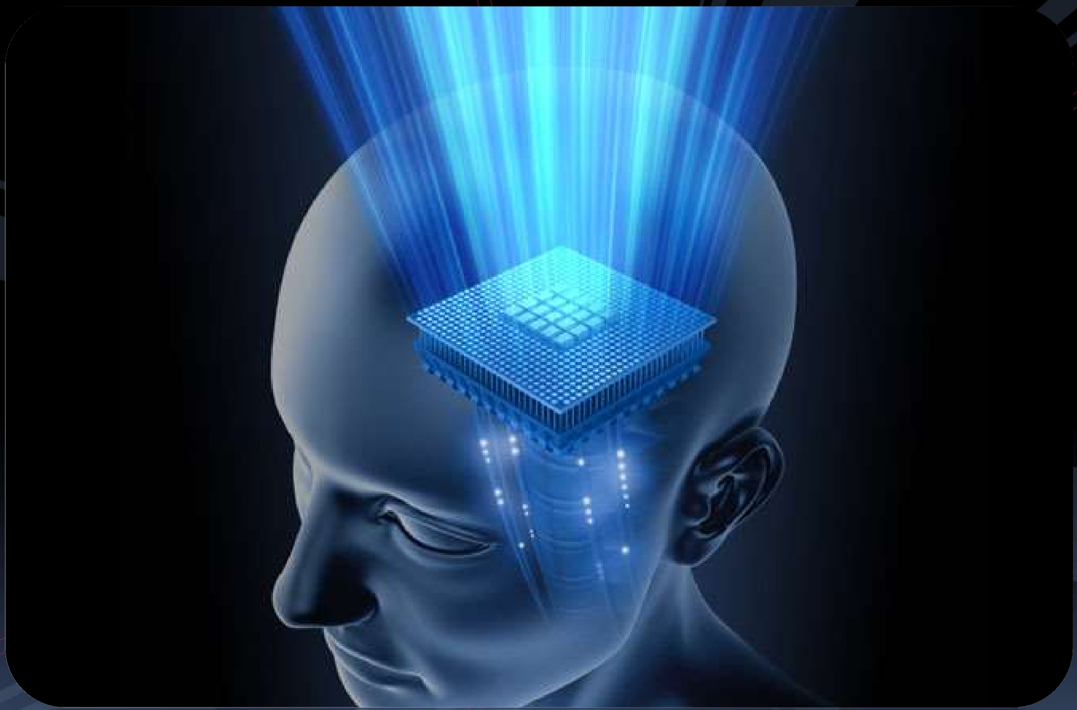
Python/Java

- Boolean variables
- Boolean operations (and, or, not, etc)
- Integers, doubles, chars
- Addition, subtraction, multiplication
- Storing values
- Executing instructions

Computer hardware

- High and low wire values
- Logic gates (AND, OR, NOT, etc)
- Registers
- Adder circuits, multiplier circuits
- Memory
- Processors

What you should already know



Programming from CSC148

- You need to have basic coding literacy.

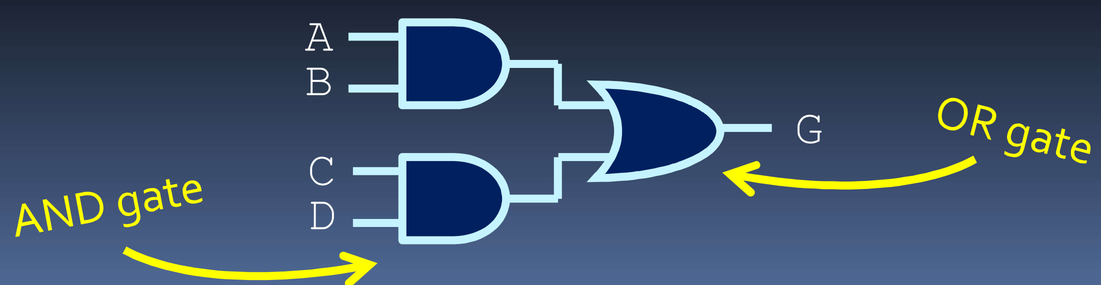
- *However…*
  - For CSC258, be prepared to let that all go.
    - Verilog → specification language
    - Assembly → low-level programming
  - Trying to connect these languages to CSC148 will only hold you back.
    - Embrace new ways of thinking.

Logic from CSC165

- Thanks to CSC165, you're already familiar with the first piece of CSC258: basics of logic gates.
- CSC165 example: Create an expression that is true if the variables `A` and `B` are true, or `C` and `D` are true.

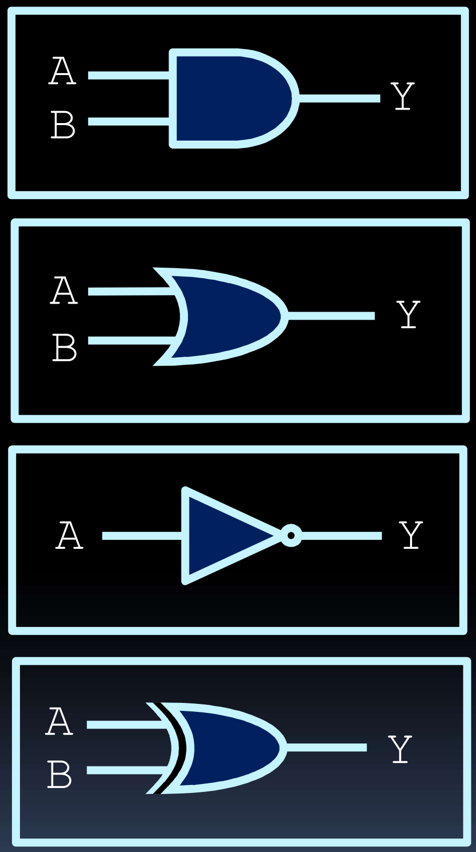
```
G = A & B | C & D
```

- CSC258 example: Create a circuit that turns on if inputs `A` and `B` are on, or inputs `C` and `D` are on:

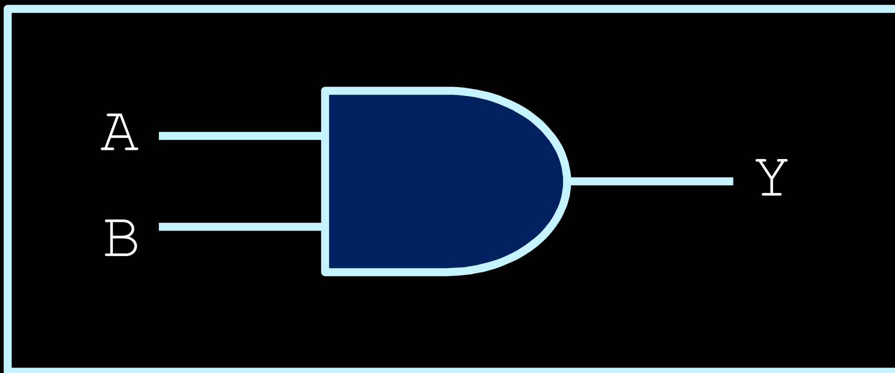


CSC165 for circuit design

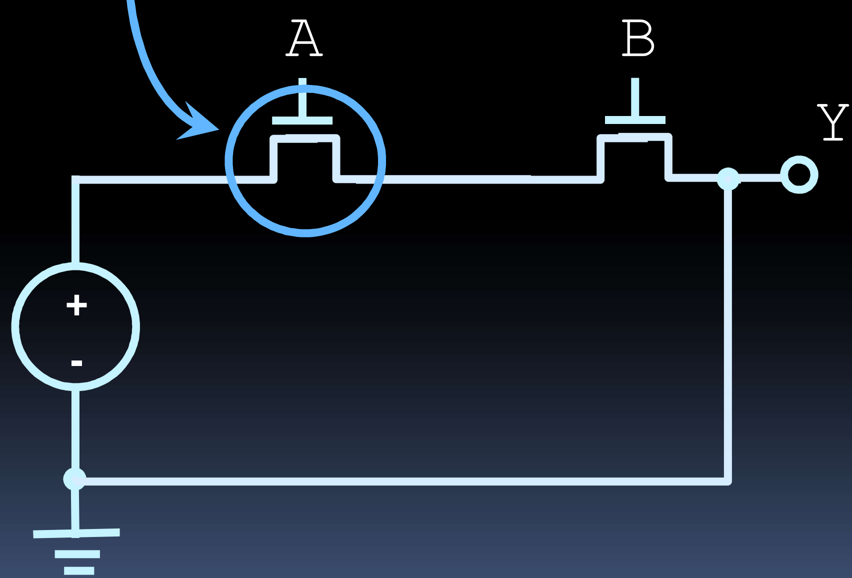
- Starting with CSC165:
  - Create simple circuits based on logical (boolean) expressions
  - Combine circuits together to create bigger and more complicated systems.
- The smallest logical operations are represented by pieces of hardware called gates.
  - Like boolean expressions, gates determine whether the ouput of a circuit will be on or off as an expression of the input signals.



AND Gates

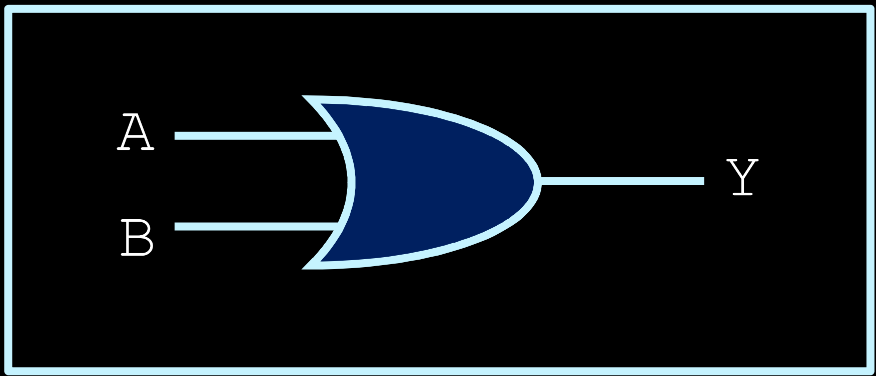


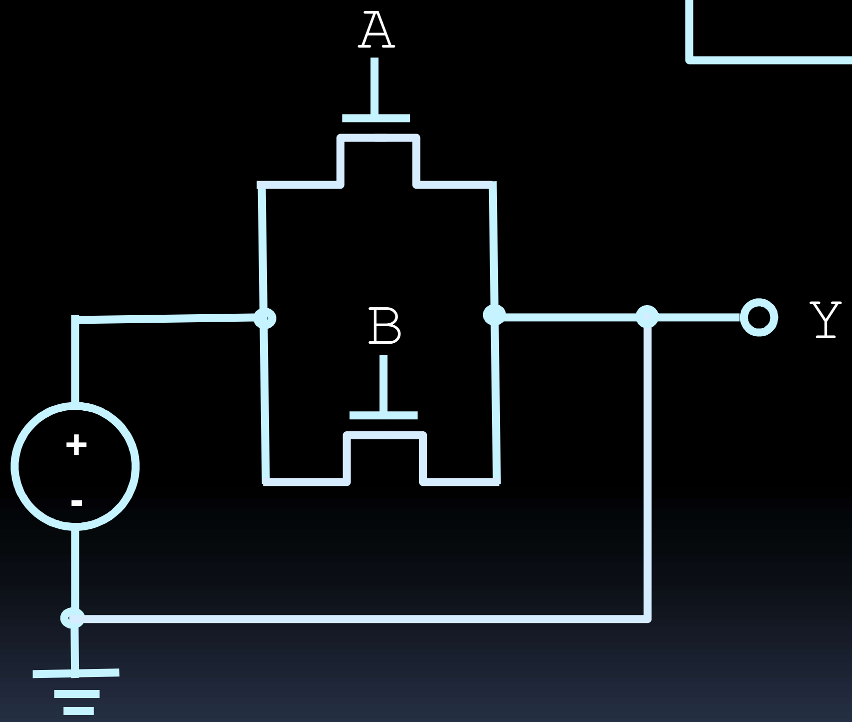
These are called transistors.
We'll learn about them shortly.
For now, think of them like switches that connect the left and right when A is turned on.



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

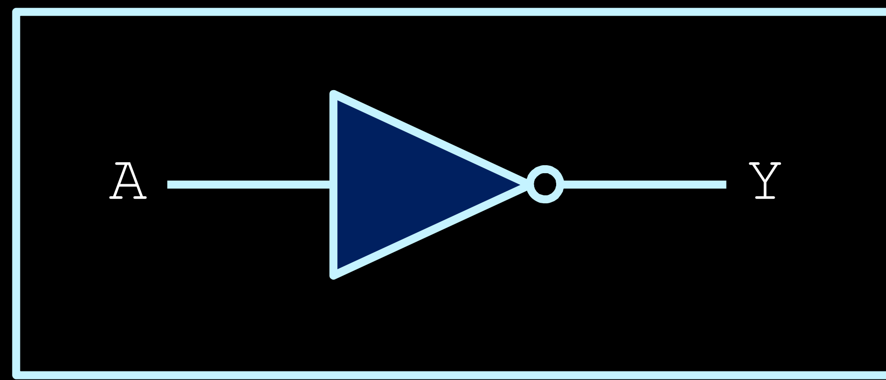
OR Gates

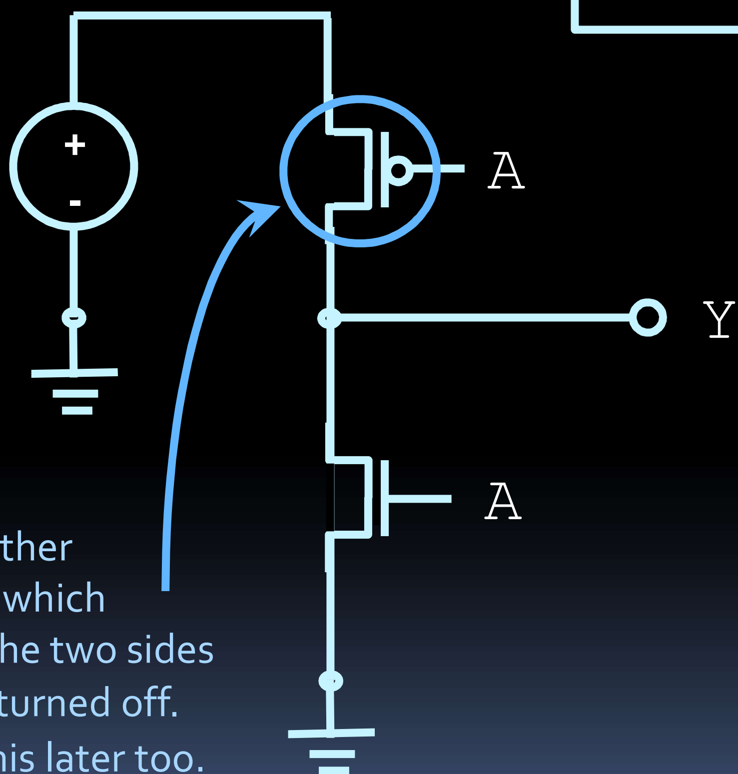




A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gates

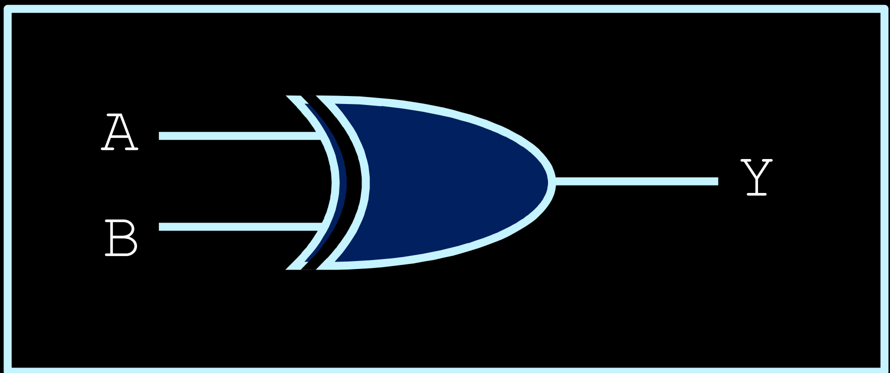


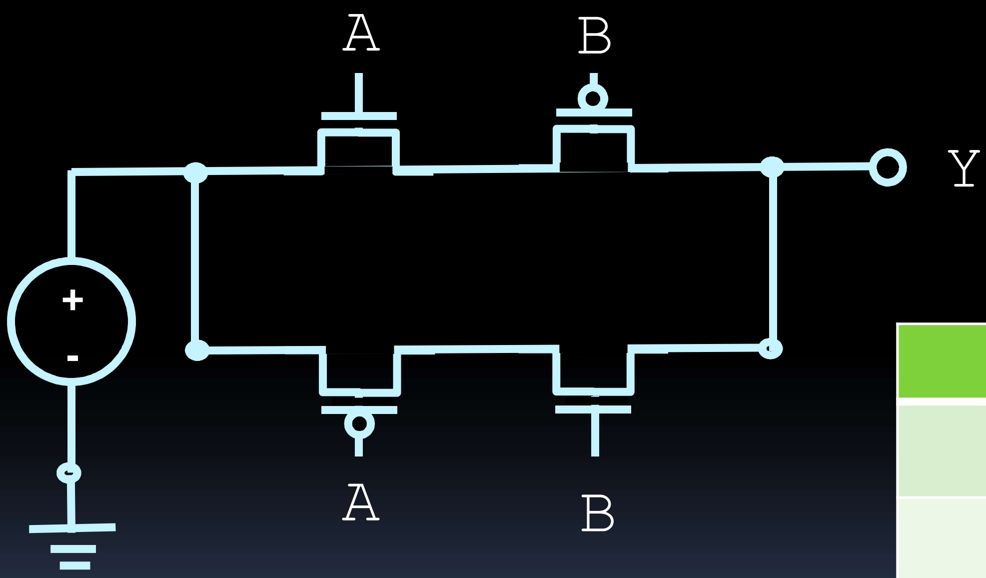


A	Y
0	1
1	0

This is another transistor, which connects the two sides when A is turned off. More on this later too.

XOR Gates



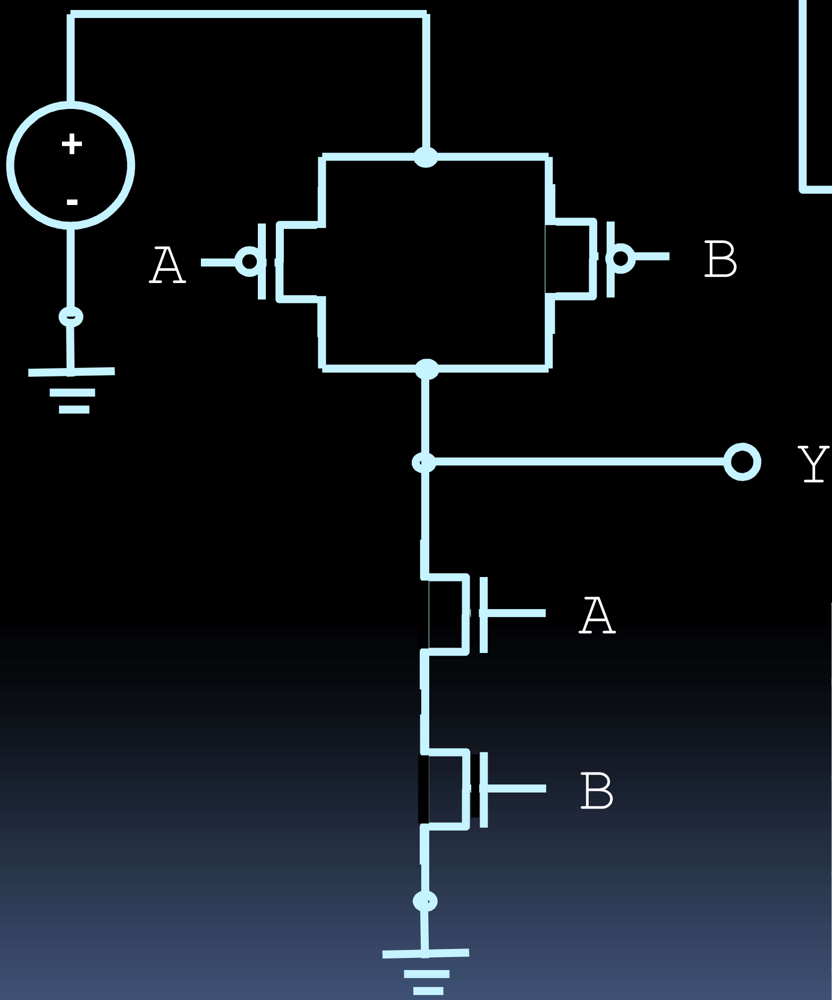


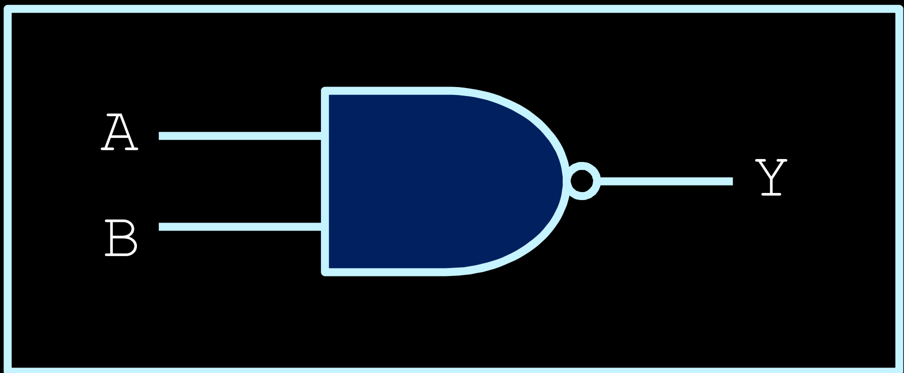
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Bill Gates

- …ha ha…moving on….

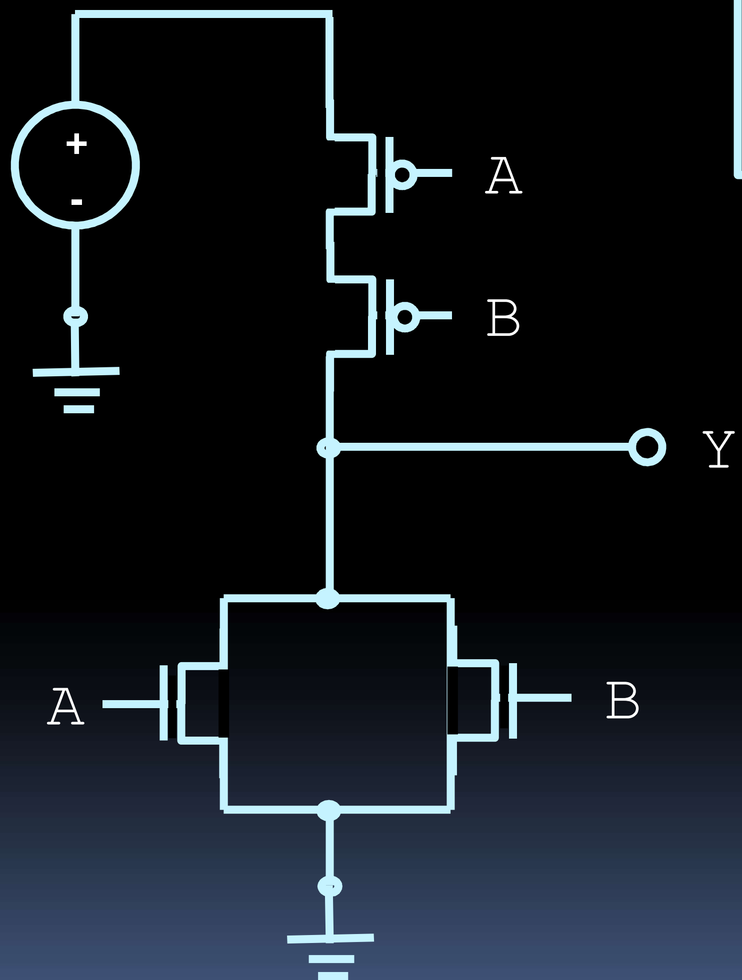
NAND Gates

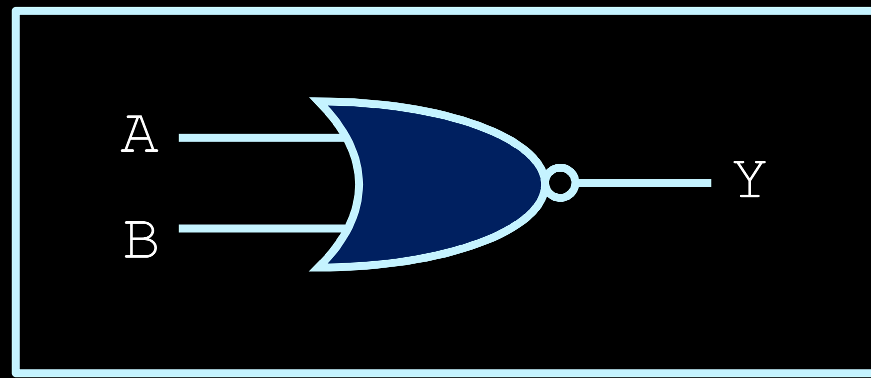




A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

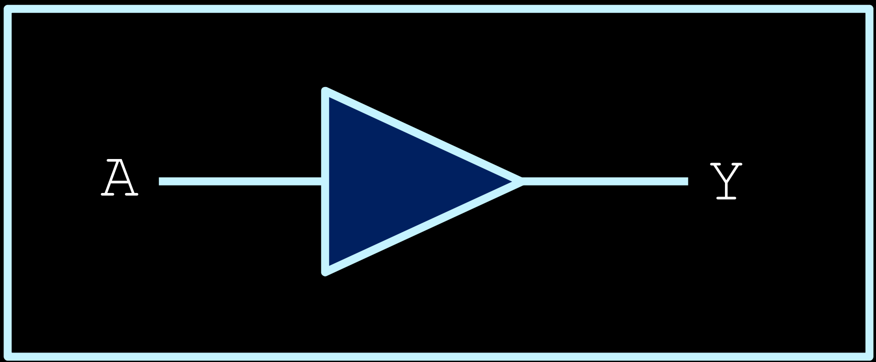
NOR Gates

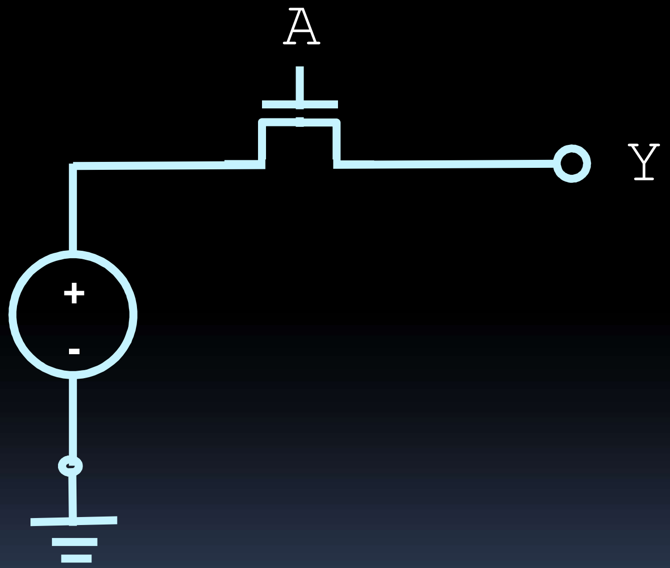




A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Buffer



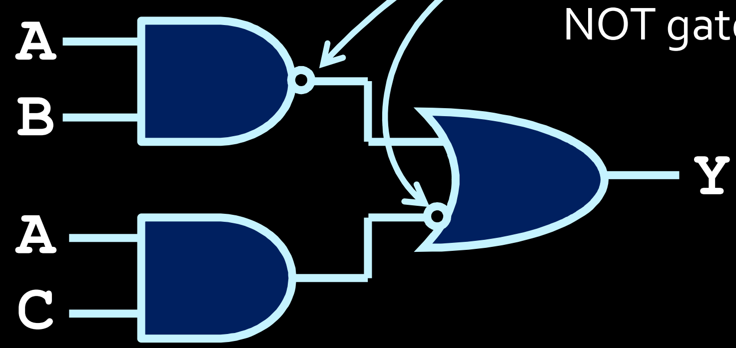


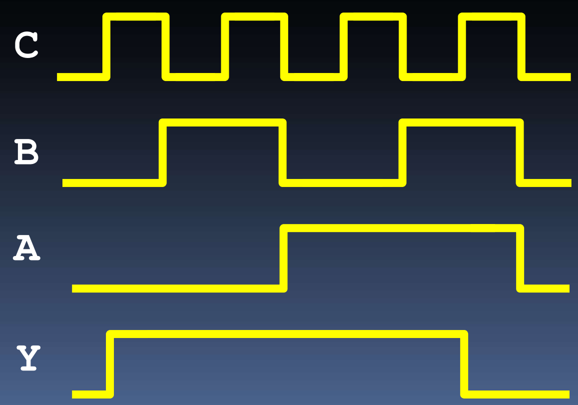
A	Y
0	0
1	1

From gates back to CSC165

(the little circle here acts like a NOT gate)

- Given a logic problem or circuit, can you make a truth table to describe its behavior (as in CSC165)?
  - Sometimes illustrated using a waveform.



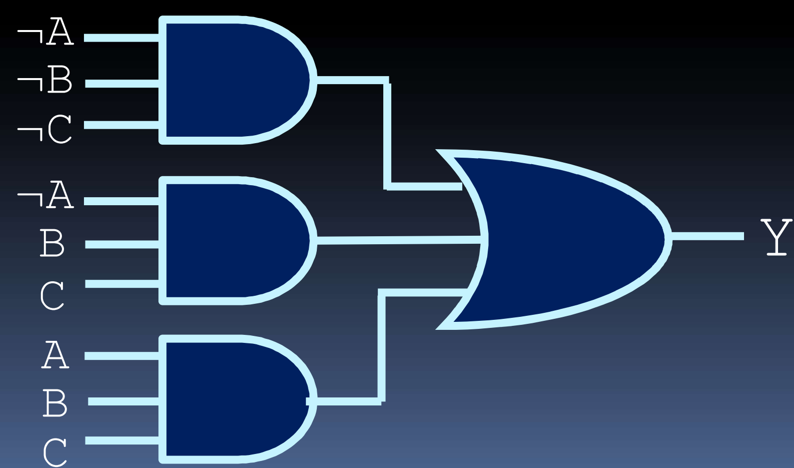


A	B	C	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

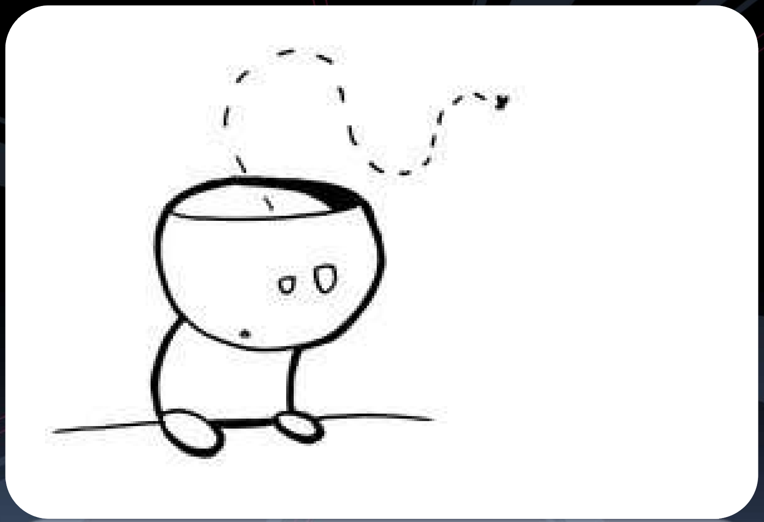
From expressions to circuits

- Creating and expressing circuit logic is similar to working with boolean logic in Python, C or Java:

```
Y = (!A and !B and !C) or (!A and B and C) or (A and B and C)
```



Things you might not know yet

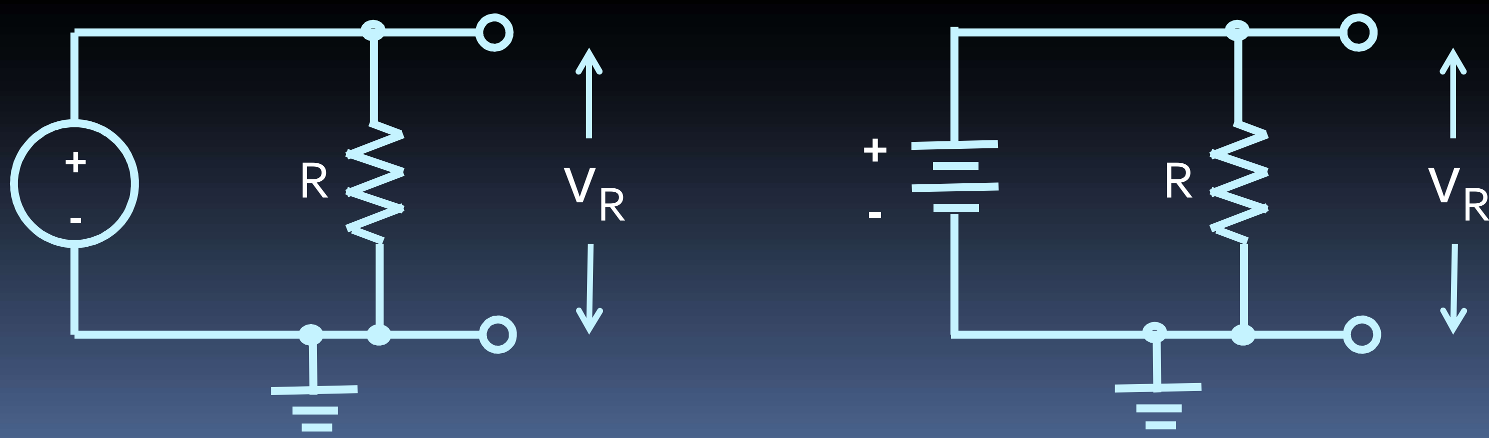


Thinking in hardware

- Although CSC258 has elements that are similar to other courses, it is very different in significant ways.
  - Unlike other software courses, CSC258 is not about creating programs and algorithms, but rather devices and machines.
    - Very important concept to grasp early in this course!
    - For instance: We need to understand what certain terms mean in the context of hardware.

“True” and “False” in CSC258

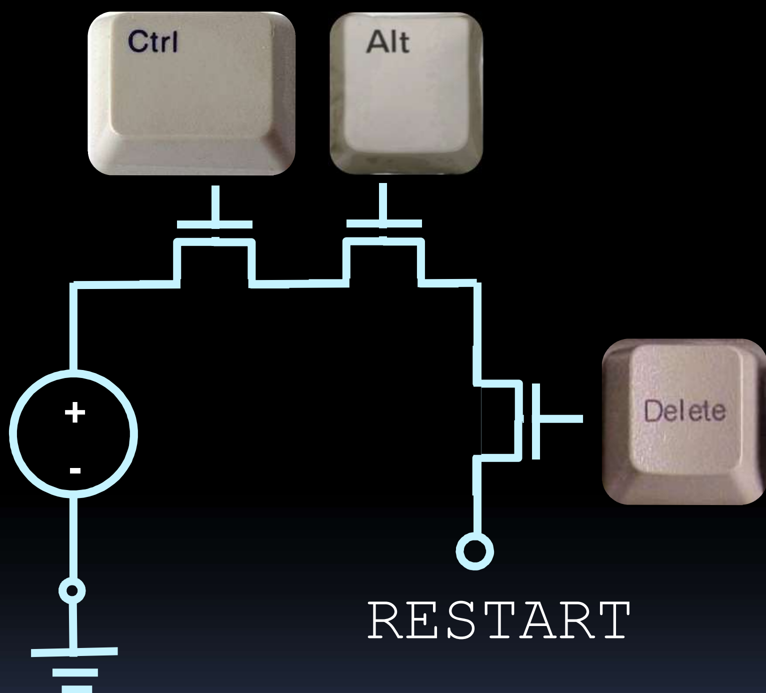
- If a circuit performs a logical operation, how does it represent “`true`” and “`false`”?
  - In hardware, these boolean values are represented as electrical voltage values on a wire:
    - “`False`” (aka zero): little to no voltage at that point.
    - “`True`” (aka one): typically a voltage difference of 5 volts, relative to the ground.



Expressions = circuits

- CSC258 tasks assume that we have input signals can be turned on (one) or off (zero), and we need outputs that combine these signals together.
  - Example #1: If the Ctrl, Alt and Delete buttons are being pressed, restart the computer.
  - Example #2: If a train is approaching the platform, only turn on the green signal light if the track is clear and the previous train is a certain distance away.
- Every electronic device uses gates to combine input signals to create these output signals.
  - Very similar to CSC165 problems, but in hardware.

Example: Ctrl-Alt-Delete



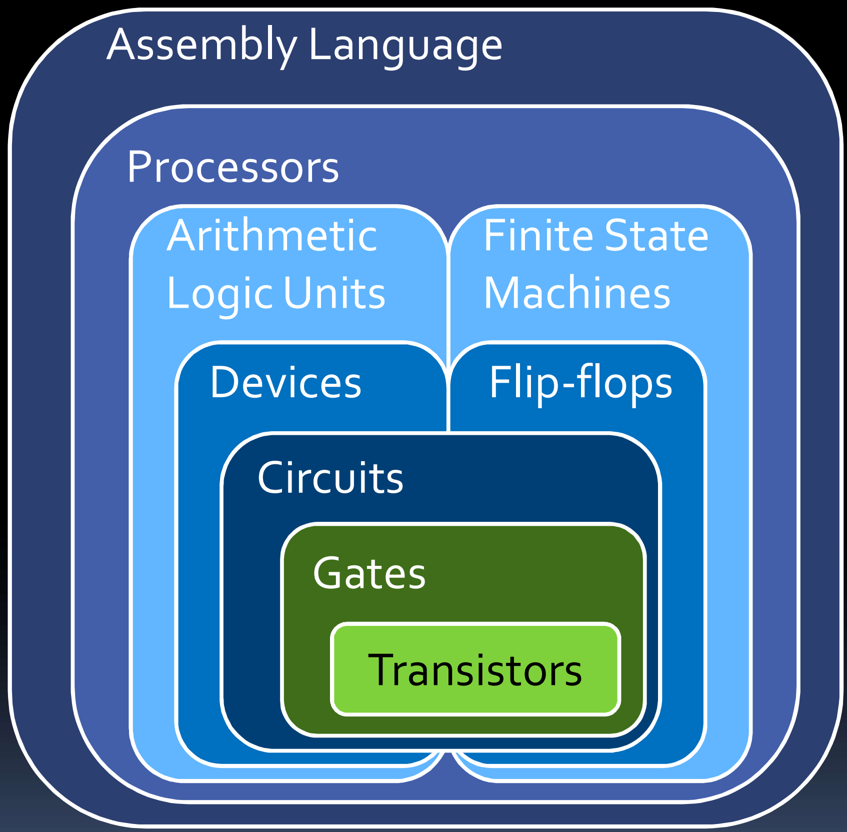
Ctrl	Alt	Del	RESTART
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

What we ask you to do (e.g. labs)

- Given a truth table or circuit description, determine the circuit that creates it.
- Look at the conditions that cause high output signals.
- Express the high conditions as a boolean statement, then convert this to gates.

A	B	C	MOVE
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

The course at a glance



Starting from the bottom

- Gates can combine values together like logical operators in C or Java.
- But how do gates work?
  - First, we need to understand electricity.
  - Then, we need to understand transistors.