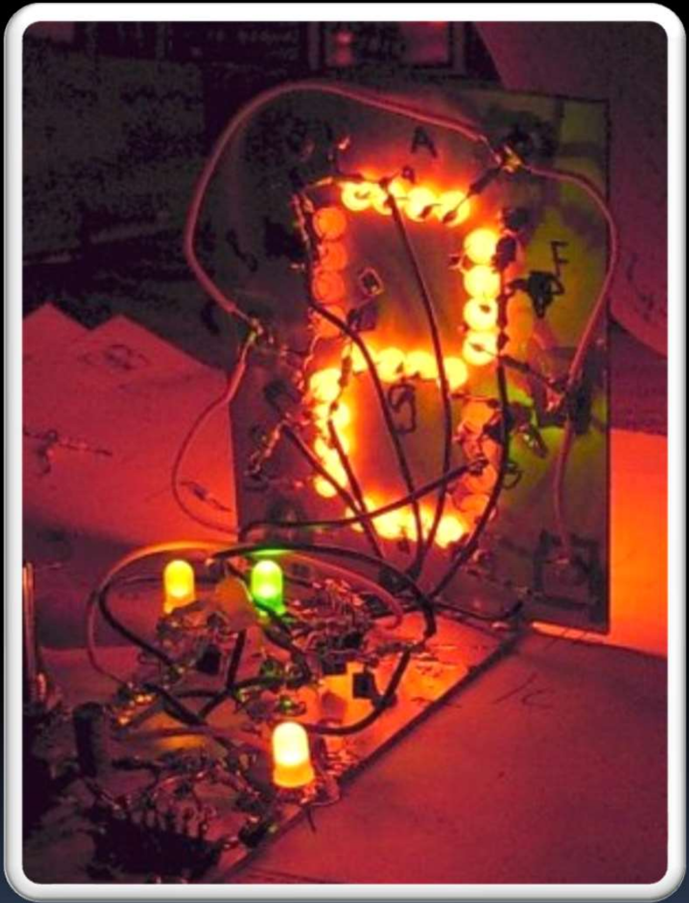
CSC258: Computer Organization

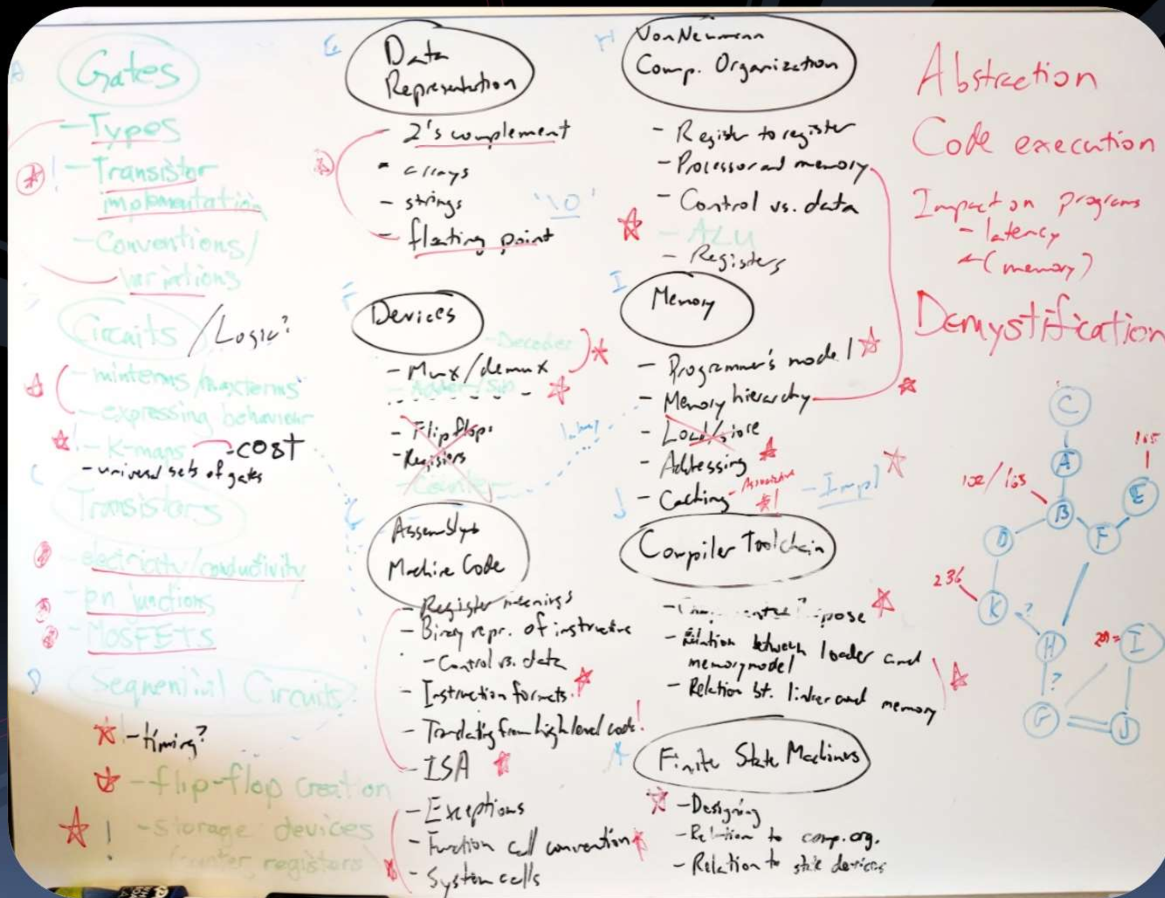
Instructors: Steve Engels, sengels@cs.toronto.edu
Ava Danialy, ava.danialy@mail.utoronto.ca

What we’re covering today

- A little about us
- What CSC258 is about
- What CSC258 involves
- Course outcomes
- Next steps



What is CSC258 about?



How does your computer work?

- To understand computer software, you need to understand computer hardware.

Limitations
Why is the maximum integer value $2^{32}-1$?

Operations
What is a pointer? How is memory allocated?

Behaviour
What is a stack overflow? How do exceptions work? What causes a blue screen error?

Example: `True` and `False`

- How does Python evaluate `true` and `false`?
  - Example: `if` statements:

```
if x:
    print 'Hello World'
    # what values of x will make this
    # print statement happen?
```

  - What if `x` is a Boolean?
  - What if `x` is an integer?
  - What if `x` is a string?

Do the answers to these questions have something in common?

True and False in Python

- Values that are treated as "false":
  - Constants defined to be "false":
    - `None` and `False`.
  - Numeric zero values:
    - `0`, `0.0`, `Decimal(0)`, `Fraction(0,1)`
  - Empty sequences and collections:
    - `''`, `()`, `[]`, `{}`, `set()`, `range(0)`

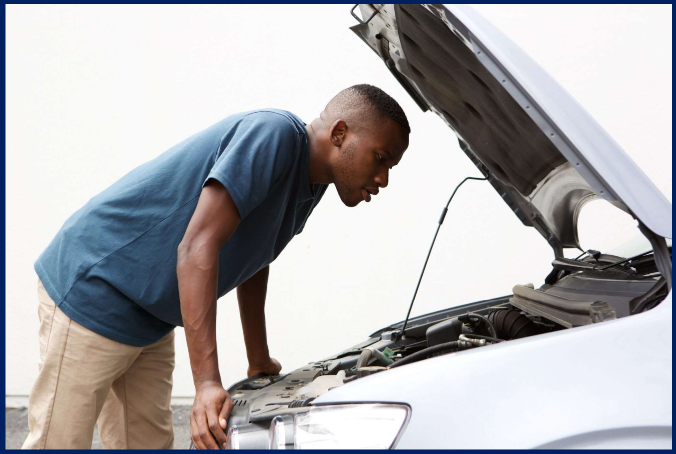
"By default, an object is considered true unless its class defines either a `__bool__()` method that returns `False` or a `__len__()` method that returns zero, when called with the object."

https://docs.python.org/3/library/stdtypes.html#truth-value-testing

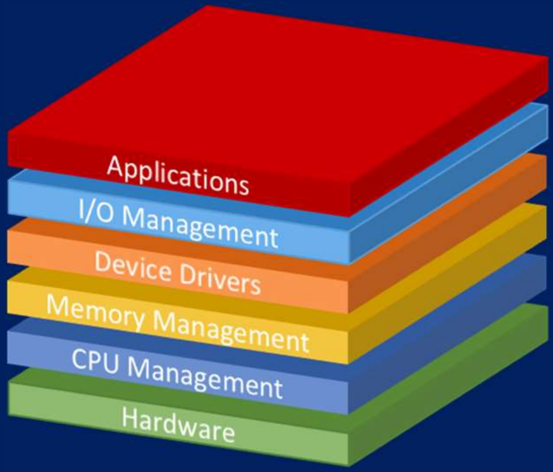
- What do these "false" values have in common?
  - They're all represented the same way in memory.
    - i.e. Zero vs not zero
  - One of the many things you learn in CSC258 ☺

Why are you taking CSC258?

Understanding the machine

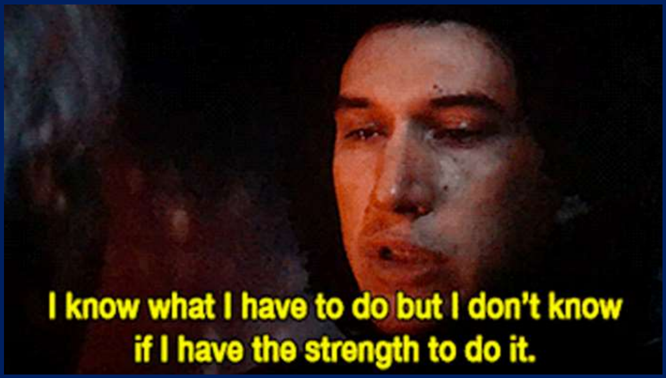


Satisfying prerequisites

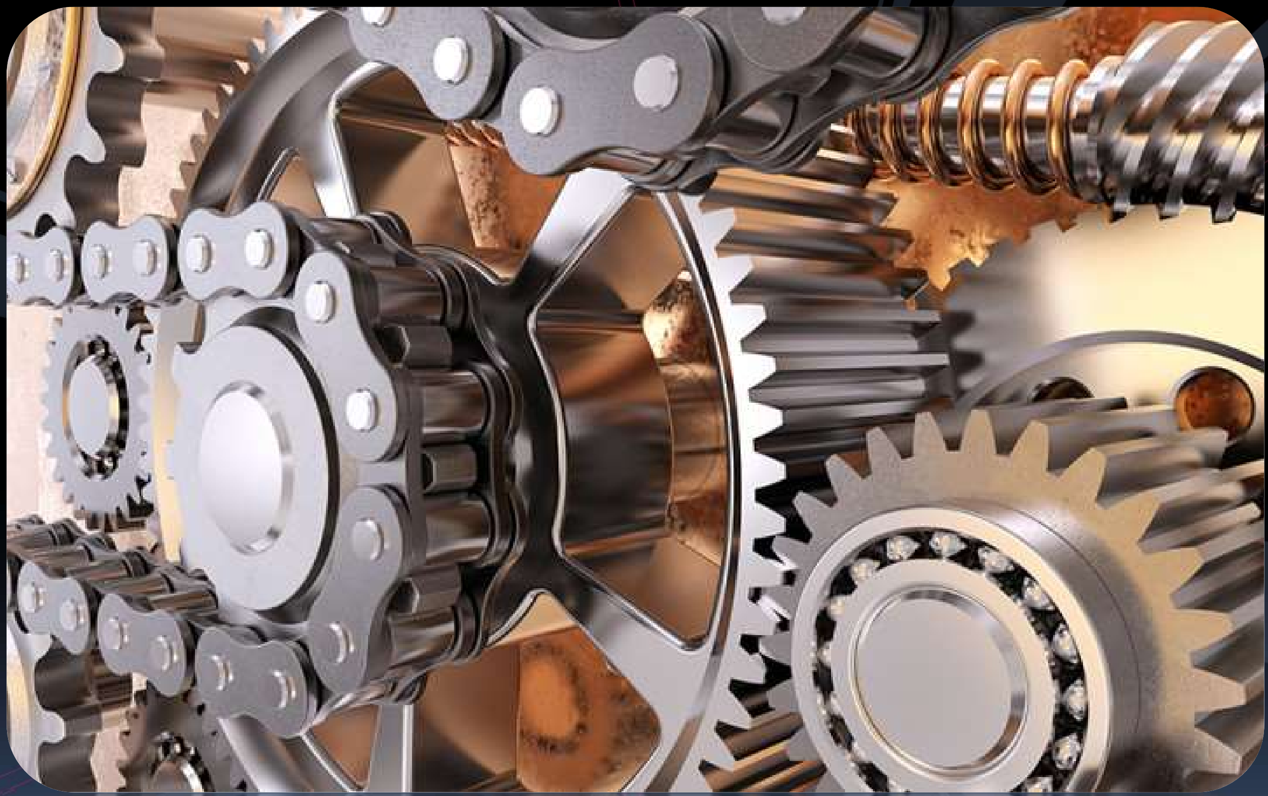


Working with devices (IoT)

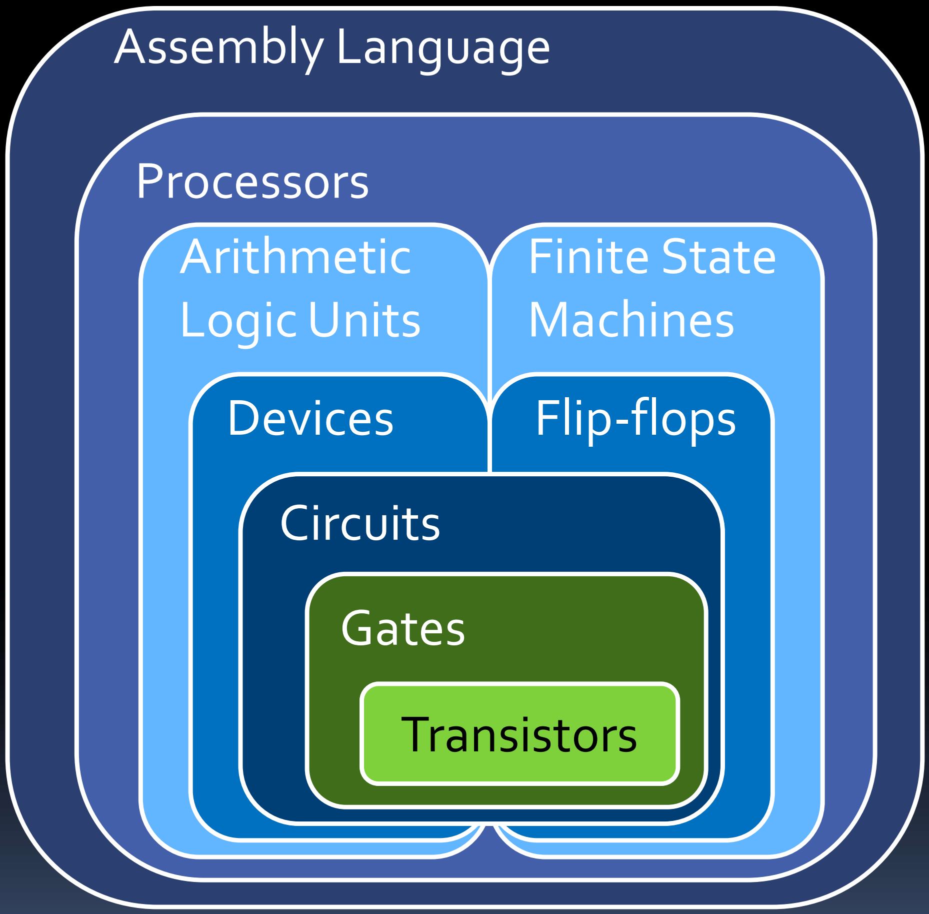
Because you have to



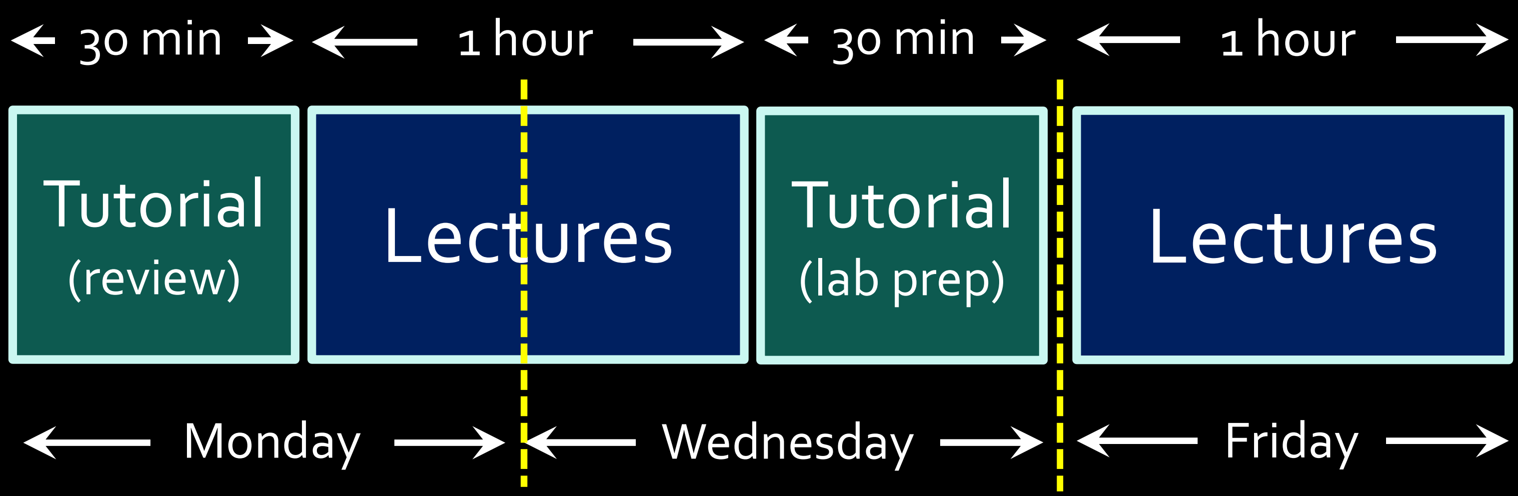
How the course works



The course at a glance



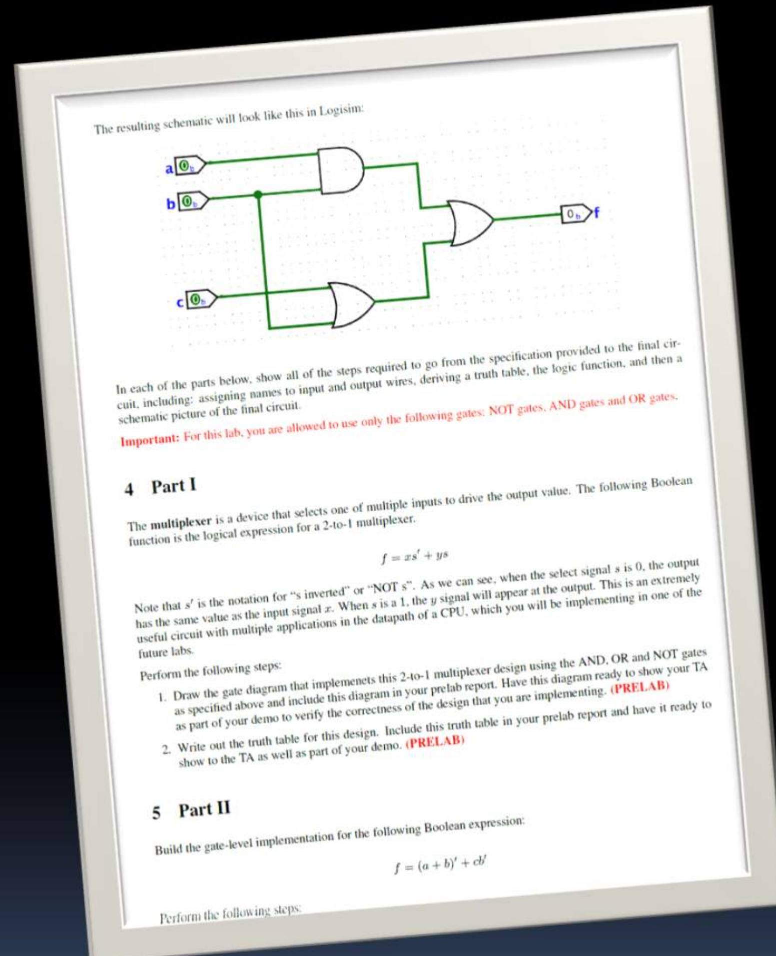
Lectures & Tutorials



- Lectures (2 hours total)
  - Lectures cover course topics (generally one per week)
  - Each week builds on the week before
  - 2022 recordings will be available on Quercus
- Tutorials
  - Monday: 30 minutes topic review (from previous week)
  - Wednesday: 30 minutes lab prep (for following week)

Lab Exercises

- Labs (28%):
  - 7 total (4% each)
    - One lab per week, starting in Week 2 (week of Jan 16^{th})
  - Each lab consists of two parts:
    - **Pre-lab → 1%**
      - Circuit creation exercises
      - Submit on Quercus before lab
    - **Demo → 3%**
      - Performed for TAs in labs
      - Minimum standard questions



The resulting schematic will look like this in Logisim:

In each of the parts below, show all of the steps required to go from the specification provided to the final circuit, including: assigning names to input and output wires, deriving a truth table, the logic function, and then a schematic picture of the final circuit.

Important: For this lab, you are allowed to use only the following gates: NOT gates, AND gates and OR gates.

4 Part I

The **multiplexer** is a device that selects one of multiple inputs to drive the output value. The following Boolean function is the logical expression for a 2-to-1 multiplexer.

$$f = xs' + ys$$

Note that s' is the notation for "s inverted" or "NOT s". As we can see, when the select signal s is 0, the output has the same value as the input signal x. When s is a 1, the y signal will appear at the output. This is an extremely useful circuit with multiple applications in the datapath of a CPU, which you will be implementing in one of the future labs.

Perform the following steps:

1. Draw the gate diagram that implemenets this 2-to-1 multiplexer design using the AND, OR and NOT gates as specified above and include this diagram in your prelab report. Have this diagram ready to show your TA as part of your demo to verify the correctness of the design that you are implementing. **(PRELAB)**
2. Write out the truth table for this design. Include this truth table in your prelab report and have it ready to show to the TA as well as part of your demo. **(PRELAB)**

5 Part II

Build the gate-level implementation for the following Boolean expression:

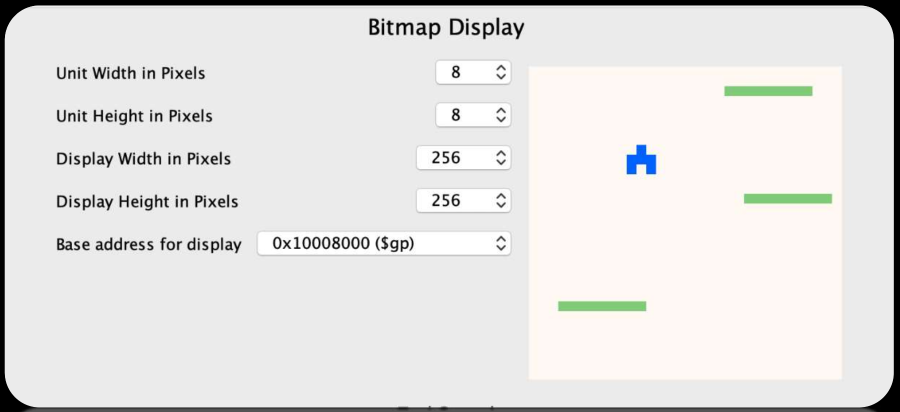
$$f = (a + b)' + cb'$$

Perform the following steps:

How do the labs work?

- Everybody has 3 hours of lecture time and a 3-hour lab time assigned to this course.
  - L0101 lab time: Wednesdays, 6pm-9pm
  - L0201 lab time: Mondays, 6pm-9pm
  - L5101 lab time: Thursdays, 6pm-9pm
- Students attend their lab session and sit at their designated station (assigned during the first lab in Week 2)
  - Pre-labs exercises are submitted before the lab,
  - Completed work is demonstrated during the lab session.
- Labs take place in BA3145, BA3155 and BA3165.
  - Students are assigned to a lab room according to last name (see Quercus to know which room you're in).

Project



- Assembly Language Project (15%):
  - Create an interactive game in MIPS assembly.
  - 5 cumulative milestones (3% each)
    - Milestones 1-3: Basic game features
    - Milestones 4-5: Advanced features
    - Can demo all 5 in the first week, if you want.
  - Milestone demos take place in the lab rooms with the TAs in the final two weeks of the course.
    - Including questions about the design process.

Midterm & Final Exam

- Midterm (19%)
  - Tentatively scheduled for Wed Mar 1, 6pm-8pm
  - Send email to the course email account if you have a conflict, along with your course schedule.

- Final Exam (38%)
  - In-person assessment (3 hours to write)
  - Must get 40% on exam to pass the course.
  - Exam date will be released midway through the semester.

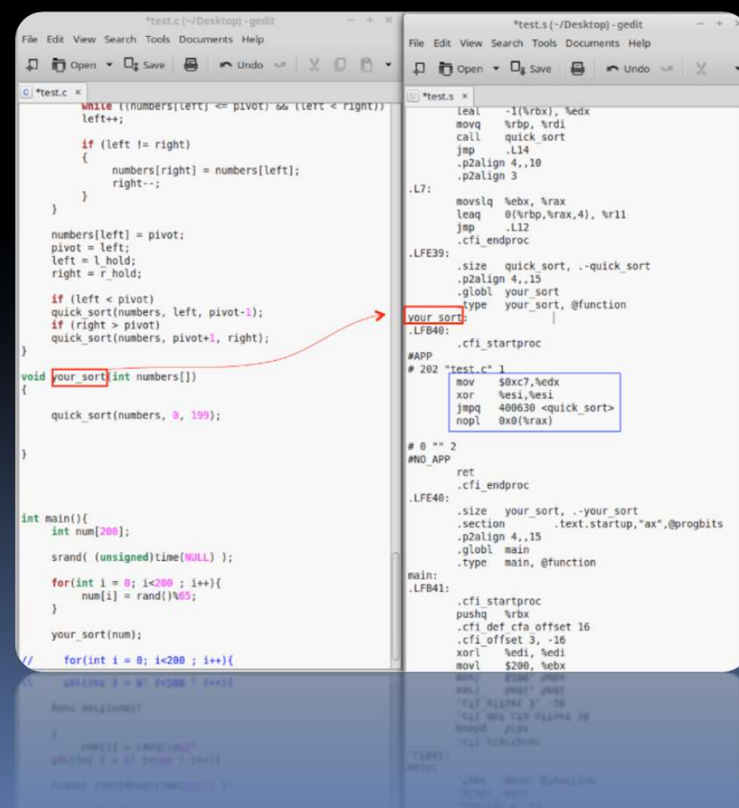
Course Objectives

Course outcomes

- Circuit creation
  - Create combinational and sequential circuits from logic gates.
  - Design circuits that implement Finite State Machines
- Microprocessor architecture
  - Implement a basic arithmetic logic unit (ALU)
  - Develop register files and memory units
  - Construct and operate the processor datapath.
- Assembly basics
  - Encode and decode microprocessor instructions
  - Translate between assembly and C programs

Building on CSC148

- Hardware knowledge helps us make sense of the software knowledge from CSC148.
  - Think of CSC258 as the prequel to CSC108 and CSC148 (or CSC110 & CSC111)
- CSC258 also complements the material in CSC209.
  - Helps you understand pointers and memory operations.



Building on CSC165 / CSC110

- Logic notation and reasoning is essential in the beginning of the course.

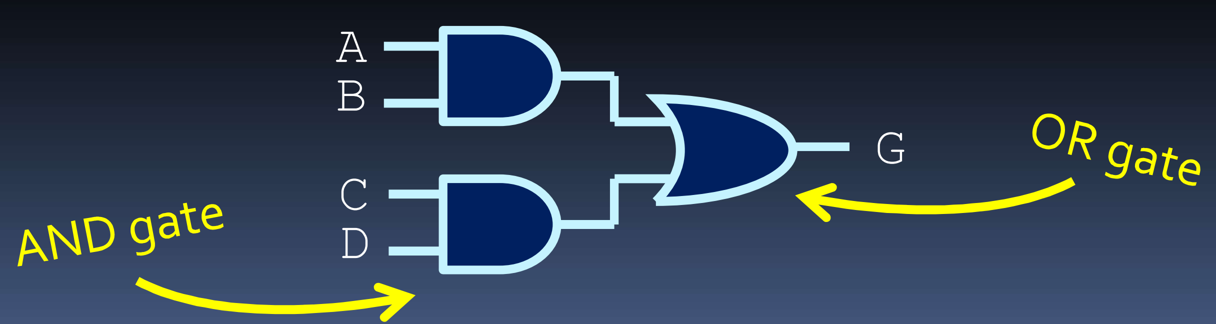
- In CSC165 (or CSC110) you use propositional logic to evaluate statements to be true or false.
- In CSC258 you create circuits whose output value evaluates to true or false, based on the input values.
  - Electrical equivalent of “true” and “false” are high voltage (5V) and low voltage (0V).
  - Also known as binary bits 1 and 0, which we see soon.

Connecting to intro courses

- <u>From CSC165:</u> Create an expression G that is true if the variables A and B are true, or C and D are true.

$$G = (A \wedge B) \vee (C \wedge D)$$

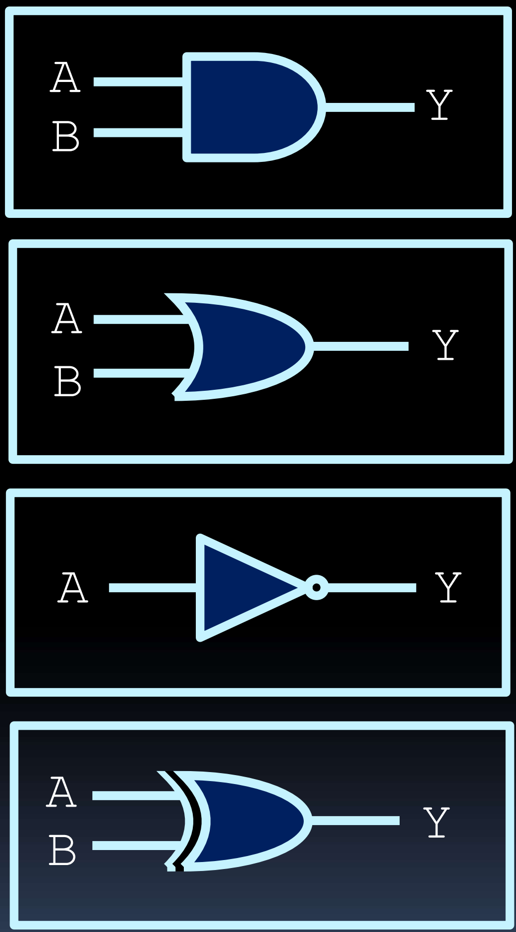
- <u>In CSC258:</u> Create a circuit that turns on if inputs A and B are on, or inputs C and D are on:



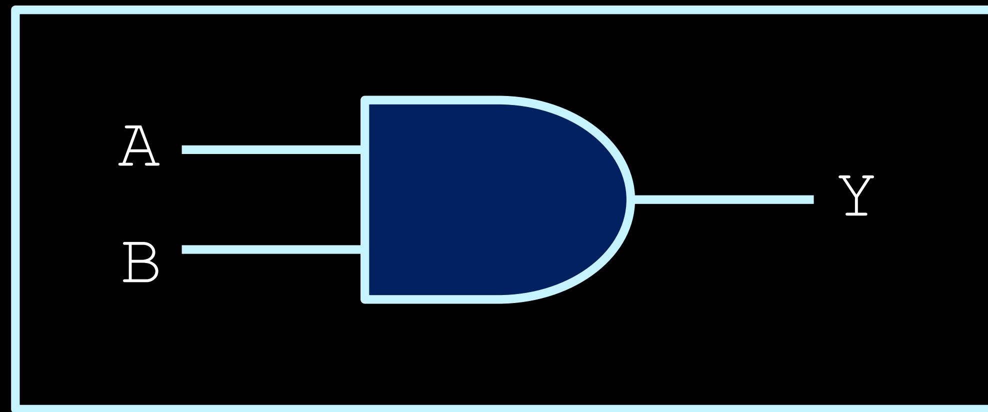
Logic Gates ↔ Operators

- Logic gates are the hardware equivalent of propositional operators in CSC165/CSC110.
  - Like Boolean expressions, gates determine whether the output of a circuit will be on or off as an expression of the input signals.

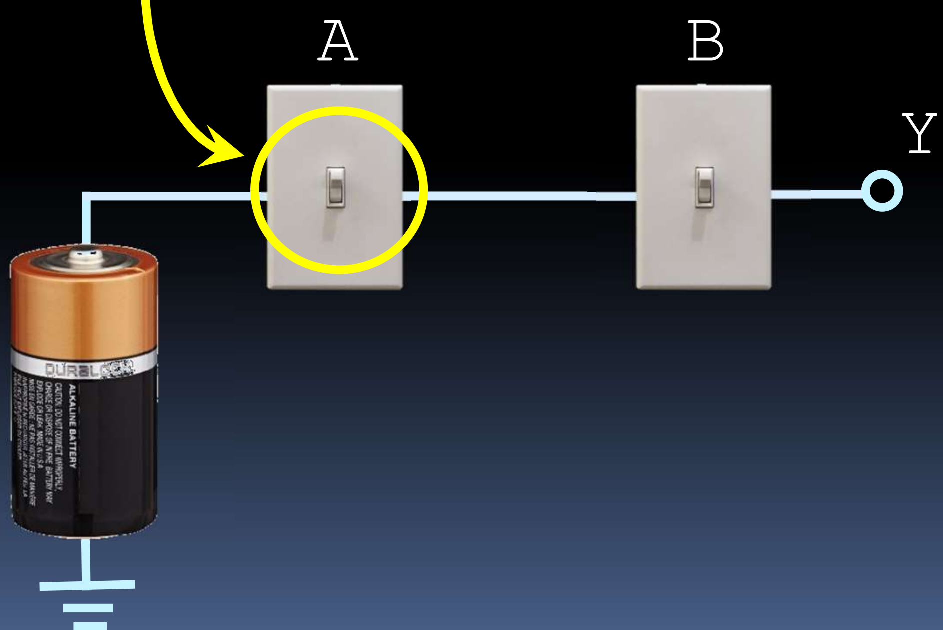
- Lab 1:
  - Create simple circuits based on logical (Boolean) expressions
  - Display truth tables that show the logical behaviour of these circuits.



AND Gates

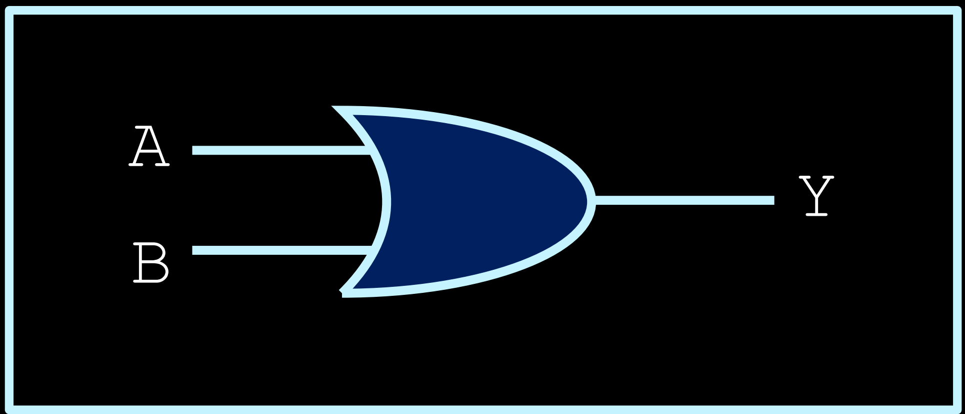


These are implemented by components called transistors. We'll learn about them shortly. For now, think of them like switches that connect the left and right when A is turned on.



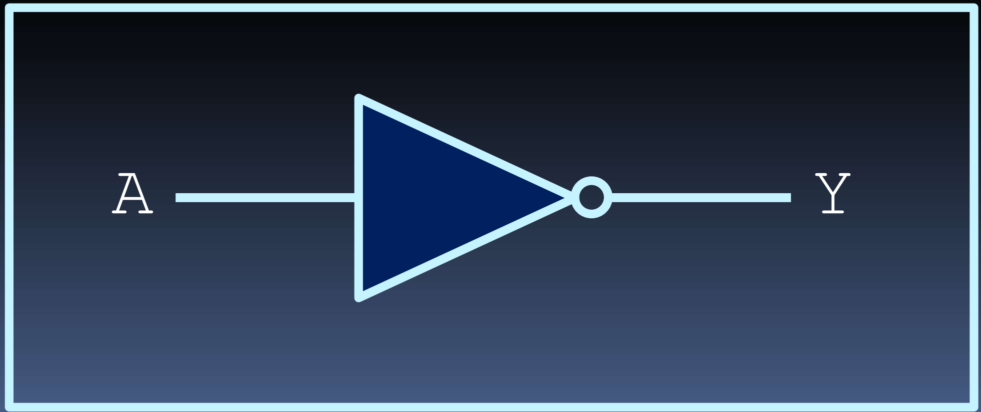
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gates



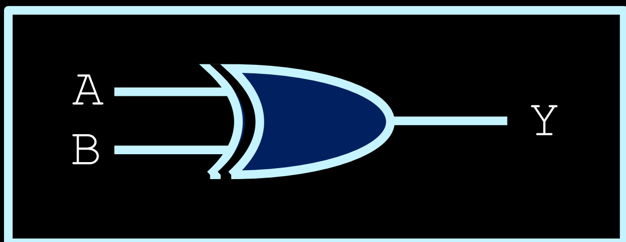
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Gates



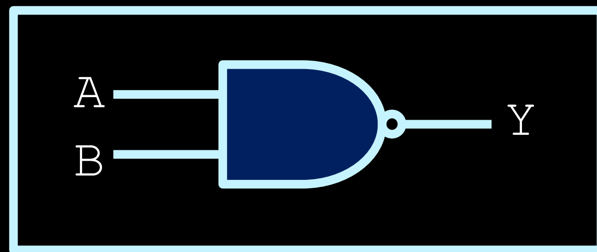
A	Y
0	1
1	0

XOR Gates



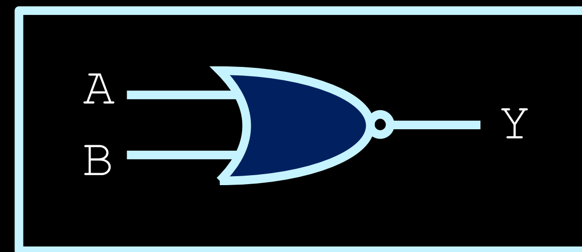
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND Gates



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

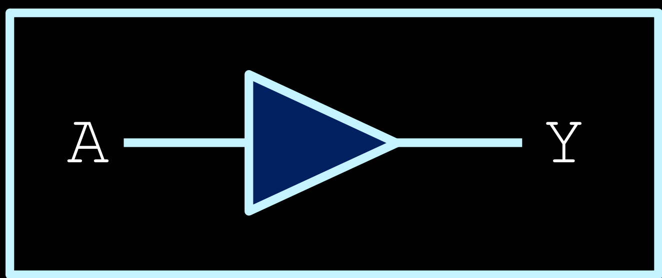
NOR Gates

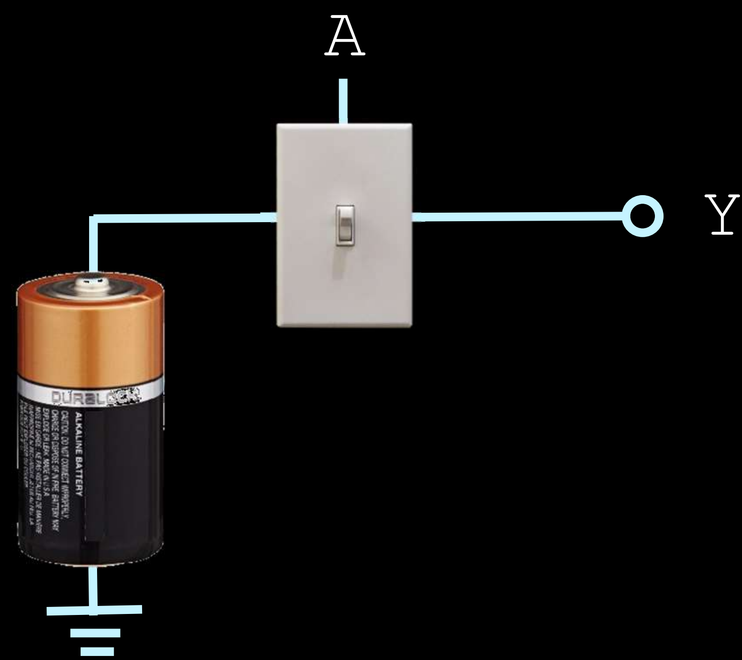


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Bill Gates

Buffer





A	**Y**
0	0
1	1

- ...more in Week 9 about what makes this gate useful.

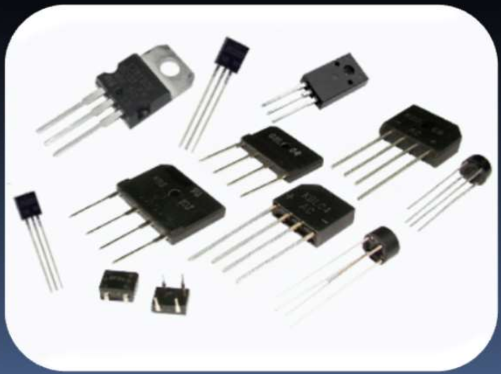
First Steps

Thinking in hardware

- Although CSC258 has elements that are similar to other courses, it is very different in significant ways.
  - Unlike our software courses, CSC258 is not about creating programs and algorithms, but rather devices and machines.
    - Very important concept to grasp early in this course!

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.
  - Finally, transistors are combined to create logic gates.



Let the learning begin