HOMEWORK 1

BINARY REPRESENTATION AND OPERATIONS

Welcome to your first homework assignment!

In the future, you will submit homework electronically on Canvas. **We will share directions about Canvas next week.**

For this week, you have an assignment but you will not be asked to submit it for grading until Week 2 when Canvas is available (week of October 25). This homework (Week 1) and next week's homework (Week 2) will both be due on Canvas on Saturday, October 31 at 11:59pm ET. **You should do the homework this week and save the answers so you can submit them next week.**

Homework 1 (for Week 1) focuses on content from Lecture 0 and Lecture 1. It includes binary representation of numbers and Boolean operations using bits. There is an optional problem about algorithms, and an optional challenge problem which introduces more complex concepts in Boolean logic. Problem 1 and Problem 2 are required and are graded. Problems 3-5 are optional.

The optional homework problems are completely optional. If you want to be more challenged or have additional practice, try them out.

Problem 1: Binary Representation

Let's practice using the binary representation of numbers, which is how a computer stores information.

a) Convert the following binary numbers to base 10:

i) 11_2

ii) 101_2

iii) 1000_2

iv) 1101_2

v) 101011_2

vi) 100110_2

vii) 11110_2

viii) 10000000_2

ix) 11111111_2

x) 11010111_2

b) Convert the following base 10 numbers to binary:

i) 7_{10}

ii) 10_{10}

iii) 33_{10}

iv) 50_{10}

v) 96_{10}

vi) 108_{10}

vii) 214_{10}

viii) 15_{10}

ix) 71_{10}

x) 146_{10}

c) Add the following binary numbers:

i) $1_2 + 1_2$

ii) $11_2 + 10_2$

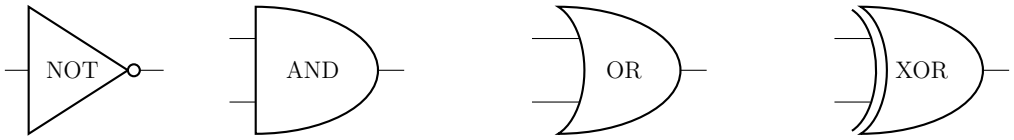
iii) $110_2 + 011_2$

iv) $1101_2 + 1011_2$

v) $00101_2 + 11110_2$

Problem 2: Boolean Logic

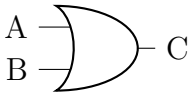
Classical computers use Boolean logic gates in order to perform various operations on bits. As a reminder, the various symbols are:



Given the Boolean logic expression, construct the corresponding truth table and logic circuit:

Example: A OR B
Answer:

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



a) A AND B

b) A XOR B

c) NOT(A OR B)

d) A XOR (NOT B)

e) (A OR B) XOR C

Problem 3: Optional Problem

Smart Traffic Intersection

A traffic intersection in a particular city consists of one major street and one minor street. The city planners have asked us to design the traffic light such that it will account for traffic on both streets. The light should only be green for the minor street if there is traffic on the minor street, but no traffic on the major street.

Use the following convention to represent the information about the intersection:

- Bit A: 0 if Major Street has no traffic, 1 if it does.
- Bit B: 0 if Minor Street has no traffic, 1 if it does.
- Bit C: 0 if Major Street has green light, 1 Minor Street has green light.

a) Construct a truth table that properly represents how the intersection should behave.

b) Draw a circuit that would implement this logic.

Problem 4: Optional Problem

Algorithm Classes

At the end of the day, we are interested in using computing to solve real world problems. In order to do this, we need to implement algorithms. The first step in solving a computing problem is deciding what type of algorithm to use.

Label each of the following situations as one of the common classes of algorithms we discussed in this lesson (shortest path, sorting/searching, or scheduling):

a) A group of students is organized by height. I want to find the student who is 5′4”.

b) A group of students needs tutors. Each student needs a tutor for a different amount of time, and a tutor can only teach one student at a time. I want to assign students and tutors so that everyone can be tutored in under five hours.

c) I am trapped in a maze and want to find my way out in the quickest amount of time. It takes me the same amount of time to travel down each branch of the maze.

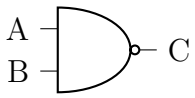
Problem 5: Optional Challenge Problem

Functional Completeness of NAND

This problem is quite challenging, and dives into the concept of "functional completeness".

We use Boolean logic gates to implement operations in computing. It turns out that with a specific set of logic gates, {NOT, OR, AND, XOR}, in combination with the COPY function, we can construct any possible operation on a set of bits. Because this can be done, this set of operations is called *functionally complete*.

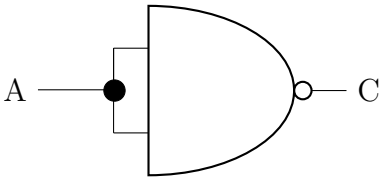
There is an additional logic gate that is commonly used when construct logic circuits that we did not discuss in detail: the NAND gate. The NAND gate comprises of an AND gate immediately followed by a NOT gate. Here is it's circuit symbol and corresponding truth table:



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

A very interesting fact is that **the NAND gate is functionally complete by itself**. This means that any conceivable operation can be performed using only NAND gates and COPY. In this problem we would like to verify the functional completeness of NAND. Our approach to do this is to show that we can recreate all of the gates in our original set {NOT, OR, AND, XOR}, which we established as a functionally complete set. If we can use only NAND and COPY to create all the gates in this set, then NAND is functionally complete. For an added challenge, you can try to do this without looking at the rest of the problem, but parts (a) - (d) go through the steps to do construct each gate.

a) First we will tackle the NOT gate. Verify that the following logic circuit produces the result NOT(A) by making a truth table.



We have created the NOT gate using NAND! this means that if we want to implement a NOT gate, we simply insert this construction.

b) Next we can create the AND gate. Doing this requires two steps:

1) Create the AND gate using NAND and NOT

2) Insert the NOT we created using NAND from part (a)

Draw the logic circuit which would implement and AND gate using only NAND.

c) The OR gate is a little bit trickier. Let's take a look at the truth tables for the NAND gate and OR gate.

NAND

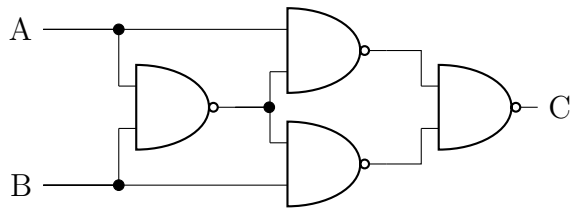
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

OR

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

They are actually very similar! If we have a NAND gate, and swap the input (A=0, B=0) and (A=1, B=1), then it will result in an OR gate. Try to implement this flipping of inputs, then draw the logic circuit which would recreate OR. Check the solutions to verify your answer. (Hint: try applying a NOT gate to A or B before they go into a NAND gate. How does this affect the truth table?)

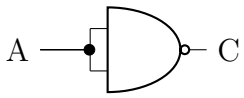
(d) Finally we must create the XOR gate. This one is really tricky, so we will show the construction. Verify that it successfully reproduces the XOR gate by making a truth table. This one is quite complex, so make sure to keep track of the input and output of each gate in the circuit.



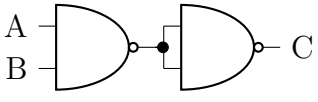
Why is this important?

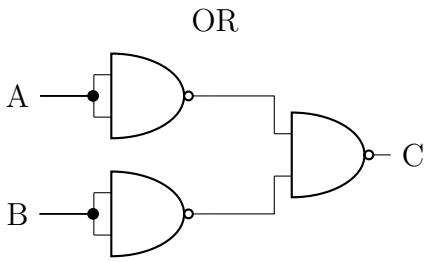
First let's summarize our results. We can create all of the other logic gates using only NAND and COPY.

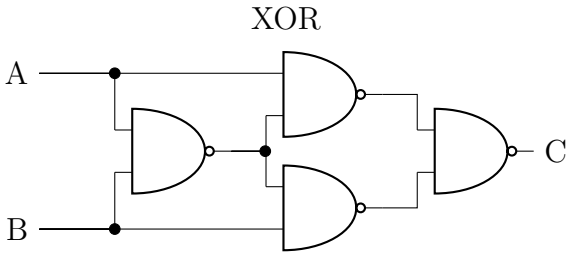
NOT



AND







The idea of functional completeness is extremely important in computing. If we want to make a computer, we must make physical circuits that perform these logical operations. If we have a functionally complete set of operations, {NAND} for example, we only need to know how to make that specific circuit, and then can make any operation necessary by combining the gates in different ways.

Once we delve deeper in quantum computing, we will see that there are parallels between classical logic gates and *quantum gates*. We will also learn what a functionally complete set of quantum gates looks like. This means that we can perform any quantum operation using those gates.