CMSC389E Project 1: Arithmetic Logic Unit- Logic Gates

Assigned **Friday February 4**

Due **Thursday February 10**

1 The First (Real) Project

Objective: Understand how to build different types of logical gates within Minecraft and what they output.

Welcome to the first project for 389E! Now, we get to finally start building our major computing circuits. It's imperative that you have completed Project 0 by this point; if you haven't, this would be a great time to talk to your instructors and let them know! They can get you up to speed.

The words of the day are **Logic Gates**. That'll be the main focus of this project, and we recommend taking a deeper look at them in case you need a review. Specifically, refer to Chapters 2.1 through 2.6 of the online textbook.

What follows is a quick reminder of the cumulative nature of projects, a conceptual overview of what our project will cover, and the project specification itself.

1.1 Projects are Cumulative

Before we start, let's just go ahead and review project policy- projects in this class are cumulative. That is, they build on top of one another. (Except for this first one) This is only natural, as we're working towards building a fully featured computer at the end of the semester.

However, as we're not sadists, we will be releasing solutions for projects after they are due- which you **can** use as a basis for **subsequent projects**. We have deliberately created this model so that if one project does not go well for you, you will not be high and dry for the next one(s). You **may not** turn in a project's solution as your solution for that same project- you may only use it for subsequent projects. Examples:

- **OK**- Building your Project 5 on top of the instructor provided Project 4 Solution and turning it in as your own Project 5 submission
- **NOT OK**- Turning in the Project 3 instructor provided Solution as your own Project 3 submission

2 Conceptual Overview: The ALU

Each project moving forward will usually have a Conceptual Overview section, where we provide a few interesting tidbits of information about the focus of the project this week. You can usually skip it if you're in a hurry, but it's a great place to get some background information on the project that you'll be building.

The **Arithmetic Logic Unit**, or **ALU** is where all the fancy math and logical operations on a computer happen, and it's housed in the CPU. In our case (and in the case of general computer architecture), this is pretty much limited to **logical & arithmetic** operations on **integers**. To perform these functions on floating point numbers, one would usually make use of a Floating Point Unit.

At the end of the day, all computers have to do math- usually a huge amount of it. Housed inside the CPUs of computers are usually multiple ALUs, all working at incredibly fast speeds to perform the functions requested of them.

If you're interested, my challenge for you is to take a look online and see where the ALU is physically located in the chip on the computer (or phone!) that you're reading this project spec on.

3 Building the Logical Portion of the ALU

In our case, we're going to start with the ALU when building our big CPU in Minecraft. We're going to split it up into a few parts, but you'll see why. Our ALU is going to be the real brains of the operation and it's going to have quite a few capabilities, but we're going to take it step by step.

The first thing that we'll be doing- the topic of this project- will be **the logical operations**, i.e. `AND`, `OR`, `NOT`, `XOR`.

First, you're going to want to load the project up. Do this using the following:

```
/test load 1
```

Before we start putting this project together for real, we're going to have to discuss **buses**. This is how we are going to handle wire management for our inputs and our outputs.

Take a look at the input and output blocks that you have available. You'll notice that you have the following inputs: `iA0`, `iA1`, `iA2` and `iB0`, `iB1`, `iB2`.

This is where all of your inputs will come in. You'll also find that you have a huge set of output blocks (sorry!). This is normal. Your goal is going to be, for each logical function in the set of `NOT`, `AND`, `OR` and `XOR`, perform the given function on the three sets of inputs, so that we can effectively have each of the logical operations done on 3-bit inputs.

Here's an example. Suppose we were building the 3 `AND` gates. You'd want to have your circuit fulfill the following logical formulae:

$$\texttt{iA0} \wedge \texttt{iB0} \rightarrow \texttt{oAND0}$$

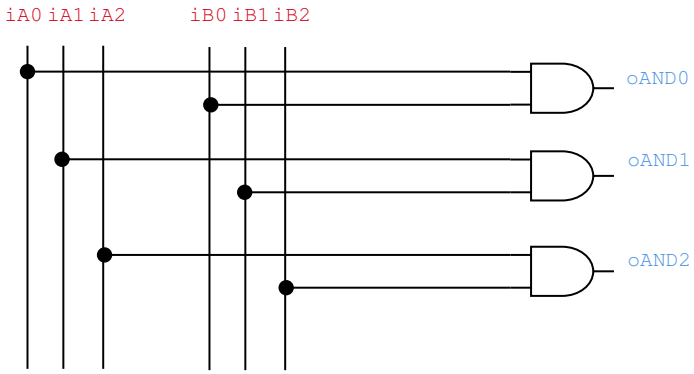
$$\texttt{iA1} \wedge \texttt{iB1} \rightarrow \texttt{oAND1}$$

$$\texttt{iA2} \wedge \texttt{iB2} \rightarrow \texttt{oAND2}$$

You'll want to fulfill these logical requirements for all of the given logic gates as well. Overall, you'll be building 3 of each gate- which, by the way, isn't just an arbitrary number. We're working towards a 3-bit computer here, so we need the ability to perform all sorts of operations on 3-bit numbers in general. Hence, the ability to take in 3 bits of input and produce 3 bits of output.

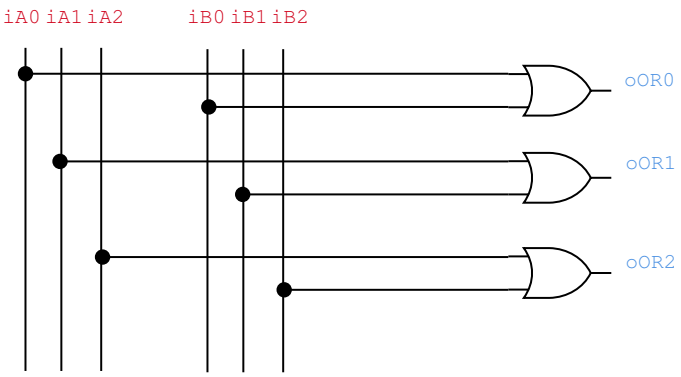
4 Schematics

Here's how this would look in a schematic, if we were to visualize this. You'll find the schematic setups for each gate as follows.

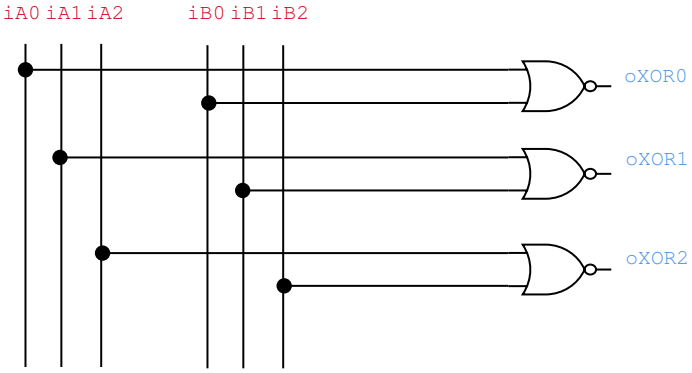


Notice how the input wires for `iA0`, `iA1`, `iA2`, `iB0`, `iB1`, `iB2` can be continued downwards. This was done on purpose. You will have to repeat this process for the `OR` and `XOR` gates below this.

Here is the diagram for `OR`. Assume that the wires for the inputs are simply continued from above. That is, this can go either above or below the circuits in the above diagram and we should be fine.



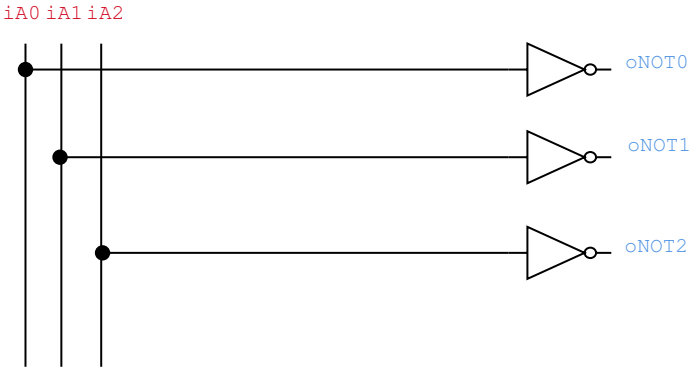
As you could guess, the diagram for what you need to build for `XOR` looks pretty much identical. Again, you will be extending the input wires downwards, so the result will be 6 very long wires (with repeaters to preserve the signal) going downwards, with all these logic gates branching off of them.



Finally, we can tackle `NOT`, which is probably the easiest of all of them. It's a bit of a special gate, as we only need to negate the input, and nothing else.

We'll be taking `iA0`, `iA1`, and `iA2`, and negating their output, then sending that into `oNOT0`, `oNOT1`, and `oNOT2`.

The diagram is below. Note that for this gate, you will not have to extend the **buses** for the `B` output wires for this gate- however, we do recommend you do, as it might help you with future projects.



You will be doing all of these one after the other, with the 6 input gates extended down all the way.

5 Submission

You will be submitting this in the same way that you did for the previous project. First, run the test start command.

```
/test start
```

Remember, the six inputs all have to be connected to the (4×3) logic gates at the same time. The tests will ensure that all your connections are hooked up correctly.

As always, attach a screencap of your tests passing, along with a zip file of your world in the assignment submission. Ask on Piazza if you have any questions!