Bradfield

BUILDING HARDWARE COMPONENTS

In this exercise you'll be building a series of hardware components (on paper/white board) using logic gates. Each section asks you to design a piece of hardware to accomplish a specific computational task out of logic gates. In so doing, you'll become more familiar with how logic gates behave, and how computers are built.

You will not construct a complete "computer" in this exercise, but you will design circuits for several core components including registers/memory, multiplexors, an adder and a subtractor. The ultimate goal of this exercise is to build the components required for a 2 function ALU (add and subtract) and a place to store the result.

For this exercise, once you've built a component from logic gates you should use it to build subsequent components. Just like you wouldn't inline a function somewhere by hand, you don't need to 'inline' the logic gates for components you already know how to build.

DOING ARITHMETIC USING LOGIC

We'd like to be able to do addition of two series of 4-bit numbers. Lets design some circuitry that will allow us to do so. Throughout this exercise you can use any components you've already built, and any of the following logic gates:

- AND
- NAND
- OR
- XOR
- NOT

Single Bit Half Adder

The first thing we're going to build a "binary half adder". A half adder takes in two bits to be added together and returns two bits of data, the "sum" and the "carry". Consider the following:

```
0+0 = 0,  sum=0, carry=0
1+0 = 1,  sum=1, carry=0
0+1 = 1,  sum=1, carry=0
1+1 = 10, sum=0, carry=1
```

Design a circuit that fulfills this function. It may help to draw the truth tables separately for (A,B,Sum) and (A,B,Carry).

Single Bit Full Adder

While we've successfully added 2 bits and gotten two relevant pieces of information, if we want to add numbers larger than 2 bits, we have to support a "carry-in". For example, consider 3+1 = 4:

```
 11 (3)
 01 (1)
----
```

```
100
```

The addition in the 1's place results in a carry. That carry must be considered in adding the 2's place, which then produces a carry which is carried into the 4's place.

```
110 (carry row)
 11 (3)
 01 (1)
----
100
```

So, we need a circuit that accepts 3 inputs: the 2 bits being added (as before) and a carry-in bit. We still only produce 2 outputs, sum and carry.

```
0 + 0 + 0 = 0,  sum=0, carry=0
1 + 0 + 0 = 1,  sum=1, carry=0
1 + 1 + 0 = 10, sum=0, carry=1
1 + 1 + 1 = 11, sum=1, carry=1
```

You may wish to draw a complete truth table. You may also use your half adder to create this circuit if you wish.

> Draw this circuit!

4-bit full "ripple carry" adder

So far we've only created an adder that supports single digit binary numbers (single bit numbers). Next, using a series of adders, draw a circuit that supports the addition of two 4-bit numbers. Once you've done it, simulate addition by determining the value on each of the input, output, and internal wires. You'll have 8 inputs (4 bits from each input) 4 outputs for the 4 sum bits, and you'll have 4 carry wires (one of which is output, 3 of which are inputs for other adders). Note that we are ignoring the first adders "carry in".

Recall that you'll now have 8 distinct inputs: 4-bits from the first number, and 4 bits from the second number. This should be explicit in your diagram. Once you've done this for 4 bits, you can easily extend the idea to n-bits.

> Draw this circuit!

A 4-bit Subtractor

We can do addition, we'd like to add support for subtraction. We've decided to support the Twos Complement format for negative numbers. Recall that in order to negate a binary number in the twos complement format, we have to first flip every bit, then add one. Using that knowledge how could we implement subtraction?

Like anything, there are multiple ways to achieve this goal. One way is by modifying a single 4-bit ripple carry adder a second way requires adding a second ripple carry adder (to do the plus 1). Using less hardware is of the utmost importance for efficiency & cost reasons; try to modify a single ripple carry adder to achieve this goal!

Confirm your solution works by simulating 4-2. In this case remember that both inputs are positive numbers: 4, and 2. Does your solution also work for 4-(-2) (four minus negative two)?

> Draw this circuit!

A 2 FUNCTION ALU

Now that we have built stand-alone components that can add and subtract, we'd like to combine them into a

single hardware component that can be "told" to add or subtract using a signal (which probably came from the machine code instruction). There are multiple ways to achieve this, but just like with the subtractor we'd like to use less hardware if possible!

Multiplexers

A multiplexor is a piece of hardware that allows us to take several different inputs, and select one of them to be output.

Build a circuit diagram for a 2-bit multiplexer. Specifically, this circuit takes in 3 bits of information two "inputs" -- we'll call them A and B -- and a "Signal" bit which says "use A" or says "use B". The MUX has a single output which is either the value from A or the value from B. For this exercise, if the signal bit is 0 we take the value from A, if the signal bit is 1 we take the value from B. Consider this truth table

```
A B S OUT
0 0 0 0
0 1 0 0
1 0 0 1
1 1 0 1
0 0 1 0
0 1 1 1
1 0 1 0
1 1 1 1
```

Because the purpose of a MUX is to ignore all but 1 input, sometimes the truth tables are collapsed:

```
A B S OUT
0 x 0 0
1 x 0 1
x 0 1 0
x 1 1 1
```

The purpose of the collapsed truth table is to make it more evident that when S is 0, B's value is irrelevant to the output (because the MUX is selecting A's value as the output).

Design a 2-way multiplexor using logic gates.

> Draw this circuit!

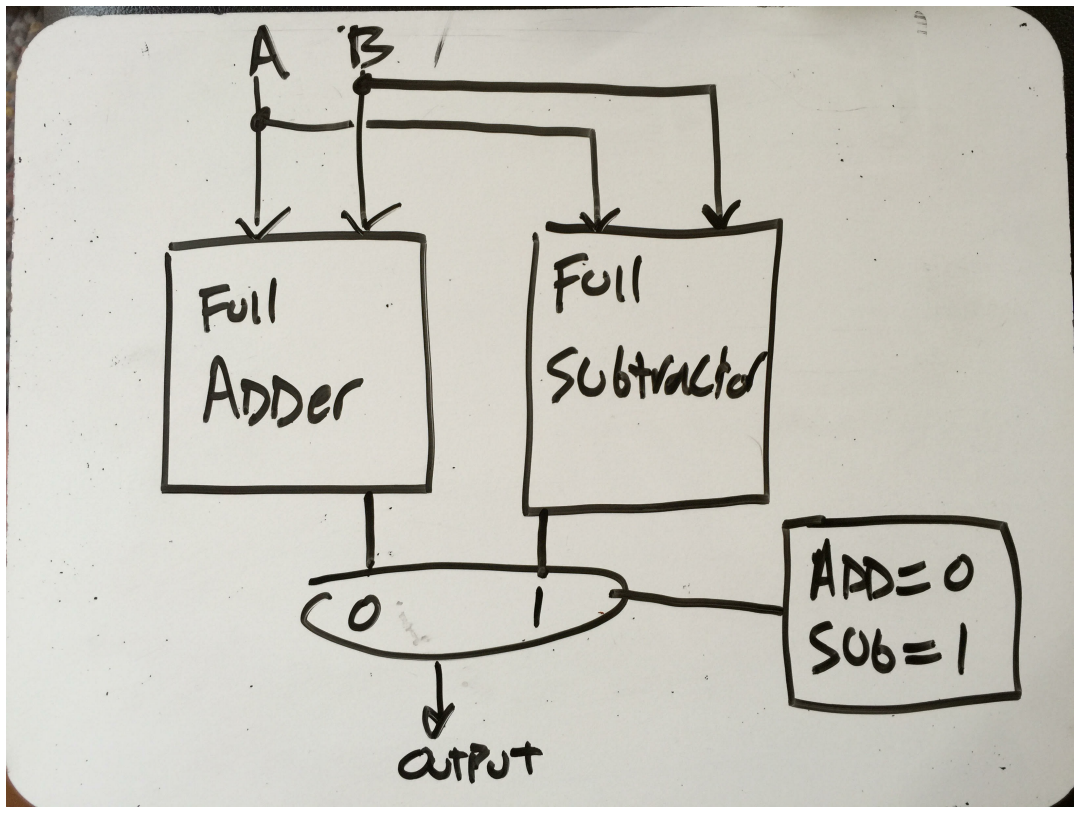
Challenge yourself to draw a 4-way MUX now.

Demultiplexors (Bonus)

A demultiplexor is a piece of hardware that allows us to take a single input and send it to one of many output channels (the rest of the output channels receive all 0's). The construction of a demultiplexor is very similar to a multiplexor. A DEMUX is not needed for our 2 function ALU, but would be needed to, for example, select a specific register to receive the output from our ALU. Consider designing one of these!

2 Function ALU

A simplistic 2 function ALU could use a single multiplexor, an adder, and a subtractor by always sending the input data to both the adder and the subtractor, then multiplex the output channel based, like this:



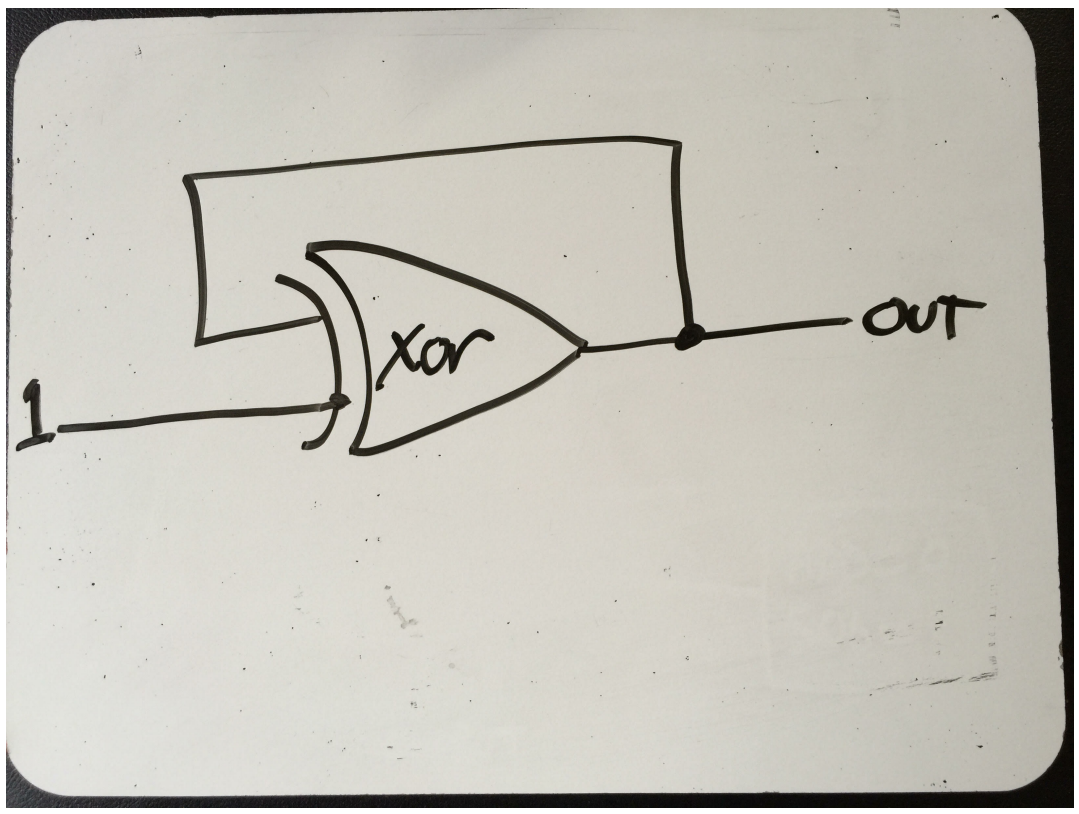
Duplicating the hardware for the adder and the subtractor, though, is costly. We can do better by adding more multiplexors (which are smaller in their construction) to cause our arithmetic-machine to selectively compute addition or subtraction, without duplicating the circuitry of our ripple carry adder. Construct such an ALU:

> Draw this circuit!

SEQUENTIAL LOGIC

So far, all the circuits we've built are "combinational" logic circuits. They accept some input, then produce some output, and they are done. This are, of course, important and useful devices for our computers. However, they aren't enough, we still need to devise circuitry that can "hold" data over time. To do this, we'll need to use "sequential" logic.

In a sequential logic circuit the output from one end of the circuit is also used as input *for the same circuit*. Consider this sequential circuit, which has an XOR gate using it's own output as input for the system, and a constant 1 as the other input.



- What is the initial value of the top input (the one that is also the output)?

- What will happen to the output value over time?

D-Latch

Design a sequential circuit that takes a two inputs D (for data), and E (for enable) and produces one output O. The behavior for this circuit is that when E is off (set to 0) the output value *cannot change* no matter what happens to D. However, when E is on (set to 1) the output value must take whatever value D currently is. If E becomes off, O must once again "latch" to the value that D was just before E turned off.

> Draw this circuit!