Jerry Lau

Lab section 4, Lab 2

2/1/2015

TAs: Daphne Gorman, William Park, Victor Ardulov

Partner: Veronica Kang

Lab 2

Introduction:

The objectives of this lab is mainly separated into three parts: memory, ALU, and putting them together.

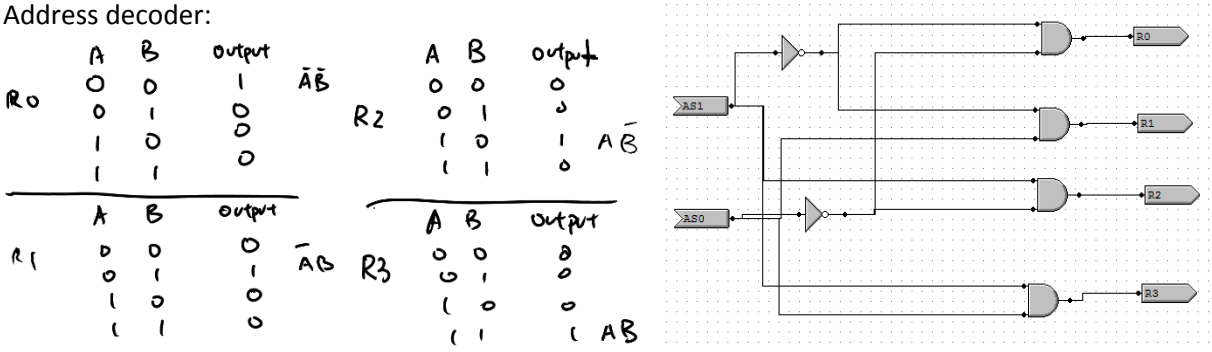
In memory, we were given the template codes. We were given the inputs, outputs, d-flip flops. D-flip flops helps to loop a bit and thus save that bit as memory. Our objective in the memory part is to create an output multiplexer and addressing select so that we could choose bit by bit on what to be sent to the display.

In ALU, meaning Arithmetic Logic Unit. We were only given two inputs, display, and some other led for indication of the status of an operation. Our objective in this part is basically creating a calculator that does multiple applications. We need to design operations that can do NOT, ADD, AND, and SUBTRACTION (extra credit).

Lastly, we have to put those two parts together. Instead of having two input keypads, we are to take advantage of the memory we created in the first part to use only one input device.

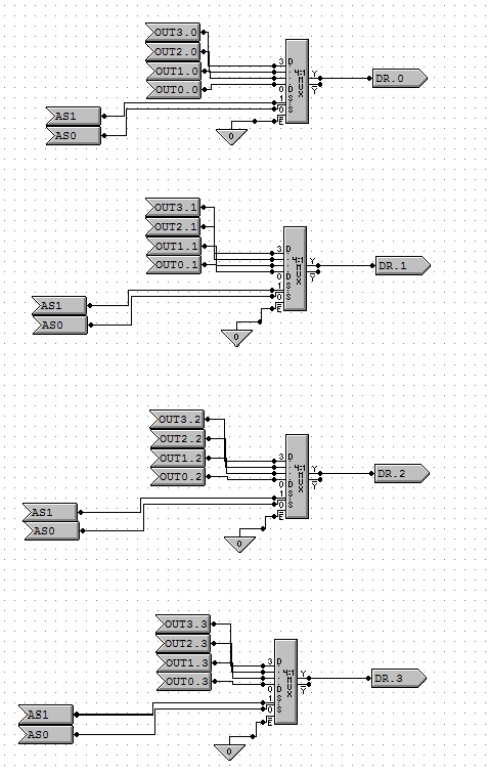
Lab Explanation:

Address decoder:



This is the first step I took to figure out what to do with address decoder, the purpose of address decoder is to help sort which bit to go to display. Using two values A(AS1) and B(AS0), we can compute all the possible combinations that we need to help display the values in registers.

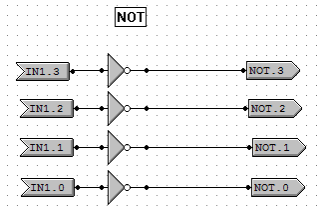
Mux (OUTPUT multiplexer):



By using the address decoder, we now can use it to sort out what specific bit we want from the registers by using a multiplexer.

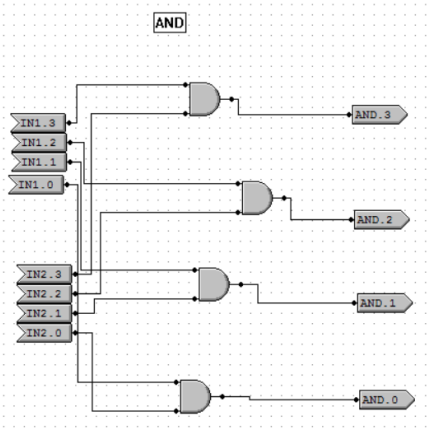
ALU (Arithmetic Logic Unit):

I started this part by creating the easy operations first.



NOT:

Well, this part is just taking any input and reverse the bits.



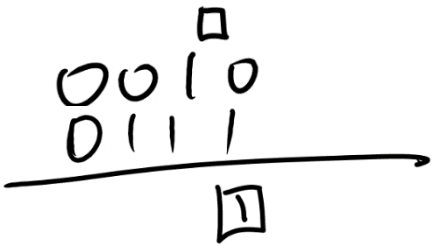
AND:

This operation is a little more. I basically made it so that input1 A value would compare to input2 A value, input1 B value would compare to input2 B value, etc.

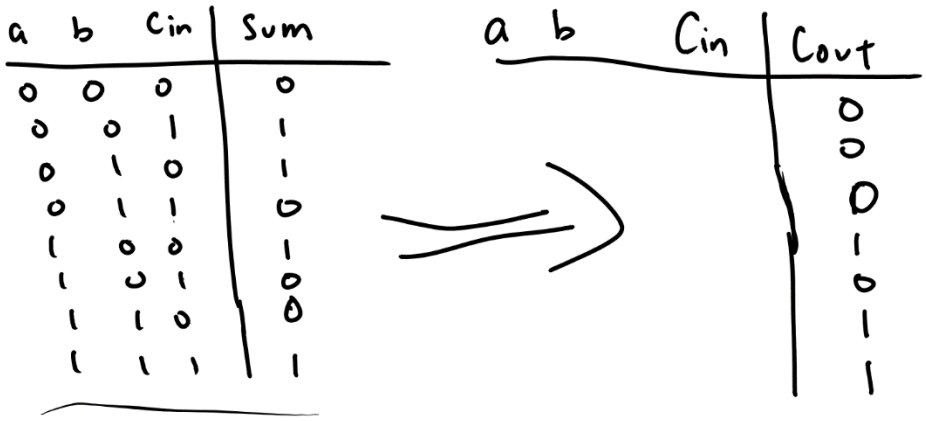
ADD:

This is the part that make this lab start to get complicated.

First I have to figure what the equation is using truth table before I could draw out the circuit. While doing the truth table, we have to consider a number called carry in.



This is how I think of carry in. The bottom is what comes out of 1+0 = 1. The upper box is the carry out, in a case where the bits are 1 + 1 = 10, then there will be a carry out of 1 carried to the next bit.



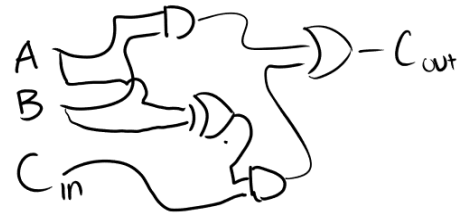
$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$$

$$= A \oplus B \oplus C$$

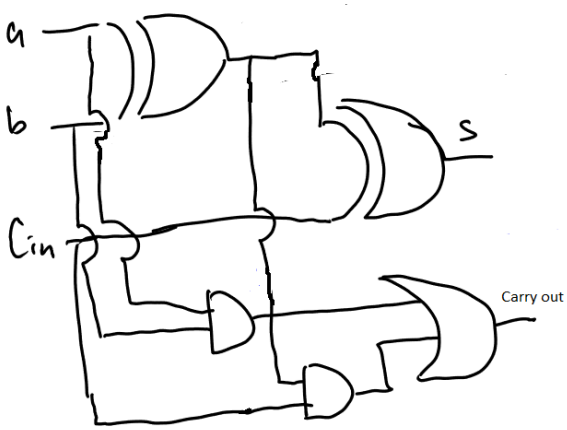
$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$

$$= AB + (A \oplus B)C_{in}$$

This is the equations we came out with and thus made the circuit below.



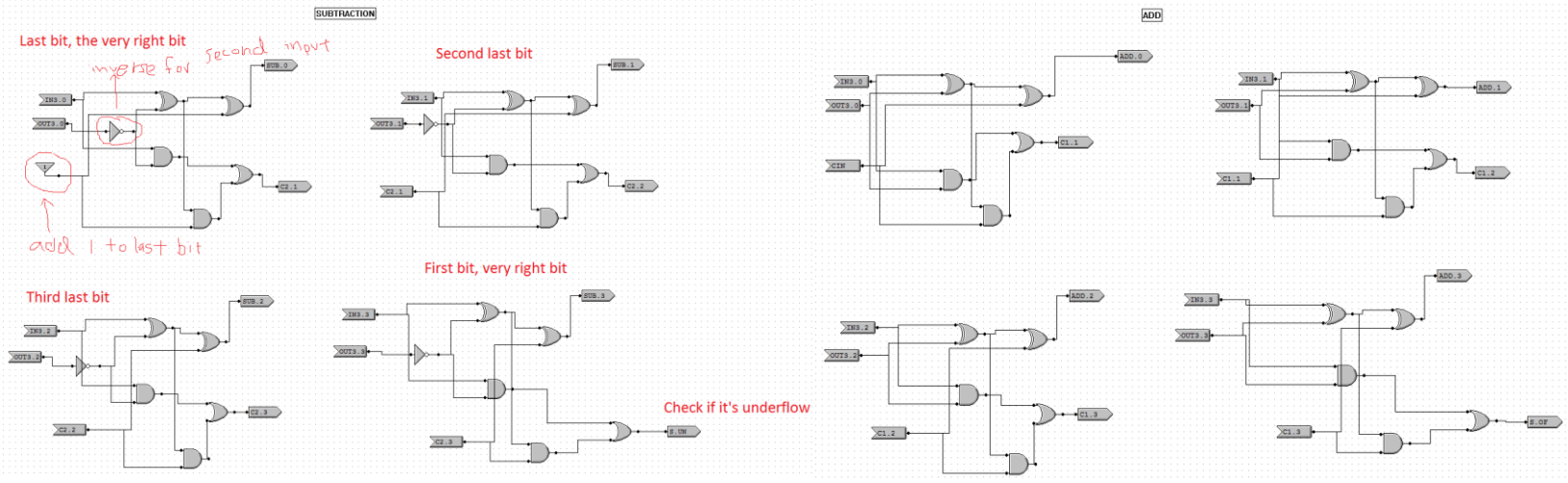
Combining these two will give us this.



The carry in for least significant bit is created by manually flipping a switch in first page to prevent any conflicting addition values. The carry out created by this circuit will then become the carry in for the next bit using the same circuit and the last carry out is linked to a led to indicate if there is an overflow. Meaning that if adding 2 4 bits value would create a fifth bit in front of the most significant bit. If yes then the overflow led would turn on if not, otherwise.

SUBTRACTION (extra credit):

This is basically the same as addition but we have to use 2's complimentary for the second input to make it work. Meaning that we have to inverse all the bits in the second output and add one to the last bit. Here, I will show a comparison.



Putting the two together:

Putting these two together is not hard, we just have to think clearly and place the flow of senders to receivers in correct order.

1. Input
2. KeyBoard/ALU (to choose which input to receive, from ALU or from KeyBoard, it's always going to be keyboard for the first time inputting because there are no previous inputs in ALU)
3. And then these bits are sent to the memory d-flip flop to get stored
4. After the memory, those bits go to Operations (go through all NOT, ADD, AND, SUBTRACTION) to compute a results.
5. The results of the operations go to OP code multiplexers and the OP decoder decide which operation is supposed to be used and shows the result on display once the button "clock" is clicked. And for BC, it becomes Subtraction because we used that slot for subtraction as extra credit.

Operation	Opcode
ADD	0001
NOT	1001
AND	0101
BC	Everything Else

I should have mentioned this in the ALU part, but here, this fills up that spot, it's easier to explain here.

6. Now these results go to two places: 1. It goes to the first page displaying an answer
   2. It goes to the d-flip flops and stores those bits

Additional information: The address select is created so that we could use 1 input keyboard instead of 2.

Result:

All the operations work, the LEDs come on when they are supposed to. I am able to switch between registers to use different values.

Discussion:

Sequential logic means using input not only from the present but also from the past. What does that mean? That means we can use previously inputs to compute our result. How do we do that? We remember it! Getting inputs and outputs latched basically means that we get those values locked up, so we could pull it out anytime to reuse them, it basically saying to store them in memory. It has very useful because without using past inputs, we can only either only have 1 result max or creating infinite number of input devices to help us compute something into a machine.

For the ALU part, we needed 2 input keyboards, result display, system status check, current operation check, an initial carry in switch, opcode select, NOT operation, AND operation, ADD operation, SUBTRACTION operation (operations consist of NOT gate, AND gate, OR gate, XOR gate, sender, receiver, LEDs), MUX.

The LC-3 has 15 opcodes.

When we are doing the lab, the opcodes only provide four slots for instructions, SUB and NAND can be created by using AND and ADD, so it would be redundant to create SUB and NAND, plus there aren't many slots for instructions in the first place. Also, SUB instruction is not needed because we can do SUB by using 2's complementary in ADD instruction.

I will simply use the bad code slot, have the same code or AND instructions but put a NOT gate before each of the sender at the end.

The purpose of N, Z, P, BC LEDs helped me keep track to see if my instructions are working fine while I was putting them together. Also, they can display a meaning that the display pad in MML cannot provide. Let say my result is a negative number, the display pad in MML can only show a single digit, it has no way showing if a number is negative or not, by using the LEDs, I could look at that number and if the LED turns on, I would know that number is not just a number but also negative.

Conclusion:

Normally, if I try to remember a word "beautiful", I will recall it simply by spelling it "beautiful". But through this lab, I realize remember something is not so simple for a computer.

In computer, each of those letters would probably be broken down into different codes and then we have to create a device to sort each number in those code one by one, have them line up the way we want, to store it in a looping device. And then we have to line them up in the right way to that each of those numbers would combine up to the right letters and then each letters combine up to the right word.

The fact that I used two lines to explain how my brain use memory and five lines to superficially explain how a computer memory works is what I learned most about this class. I am quite surprised on how sophisticated a computer's memory works, and I have been starting to try to understand everyday technology in a binary way. For example, looking at a monitor, I would think how many different switches are being turned on and off to change the color of one spot and create meaning human can read, etc. This class is quite fun.