Quinn Roemer

Engineering – 303

Lab 13

5/1/2017

Introduction/Description

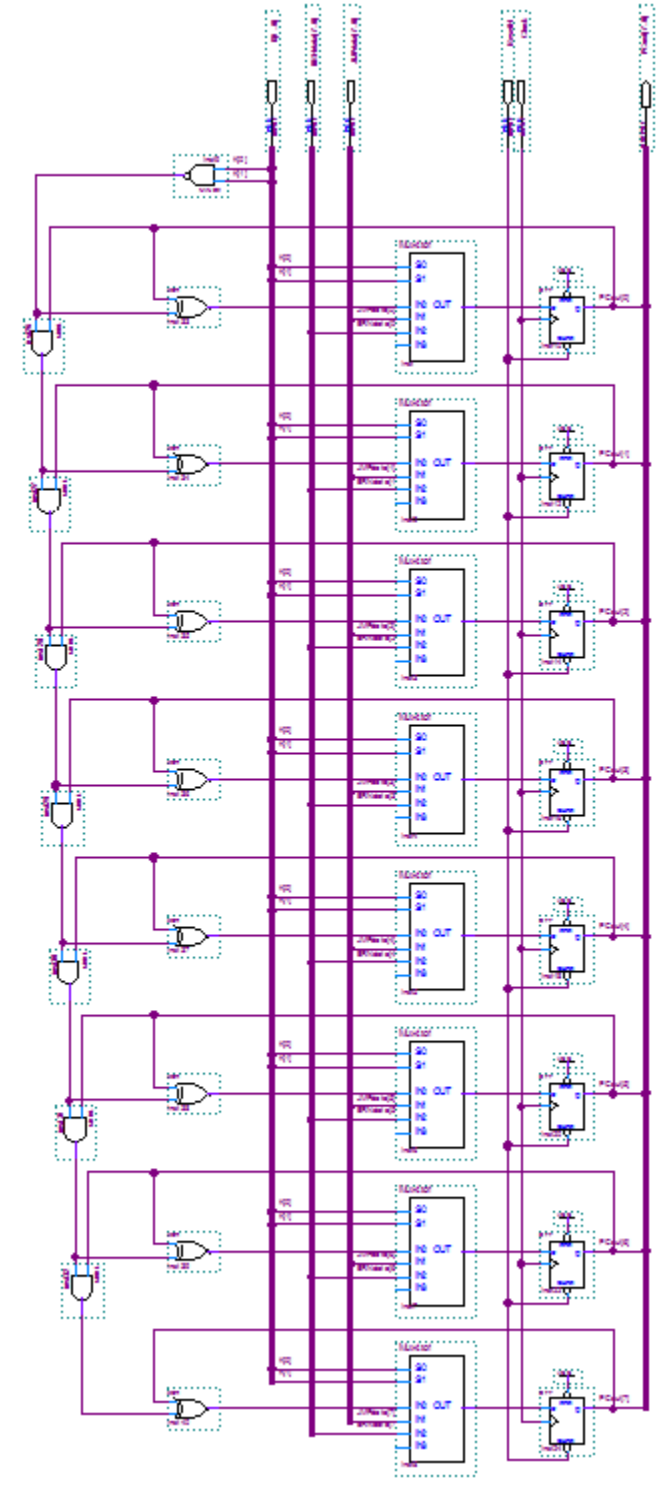
The goal of this lab was to implement several designs that were still needed before we put all of the parts together to form our single cycle computer. In this lab, I created a total of five circuits. The first of which was called the Program Counter. Also in this lab, I created a Zero Filler circuit. An Extender Circuit, Branch Control Module, and finally an Instruction Decoder.

Design

Part 0 – Program Counter

The first circuit that I implemented in this lab was called the Program Counter. This circuit needed to be capable of performing three tasks depending on the 2-bit selection signal. If the signal was 00 this circuit would simply count up in values of 1. If the selection signal was 01 this circuit would load a new value from bus A. If the selection signal was 10 this circuit would load a new value from the Extend Module. Please note, if the selection signal was 11 this circuit will fail to deliver any sensible output.

Here is the Block-Diagram for the Program Counter.



Part 1 – Branch Control Module.

The next circuit in the lab was called the Branch Control Module. This circuit essentially takes a 5-bit input and converts it into a 2-bit selection signal that can then be used to control the Program Counter. Please note, this circuit was designed in Verilog.

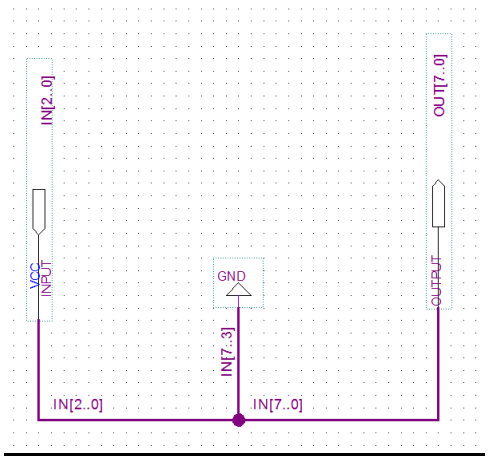
Here is the Verilog Design File for the Branch Control Module.

```
1   //A Verilog circuit for the Branch Control Module
2
3   module BranchControlModule (PL, JB, BC, Z, N, K);
4
5   input PL, JB, BC, Z, N; //5 single bit inputs.
6   output [1:0] K; //A 2 bit bus.
7
8   assign K[1] = PL & ~JB & ~BC & Z | PL & ~JB & BC & N;
9   assign K[0] = PL & JB;
10
11  endmodule
```

Part 2 – Zero Fill

This simple circuit is called Zero fill. It performs the simple but all-important task of converting a 3-bit input into an 8-bit output by adding binary zeros to the left of the original number. Please note, while trying to compile this circuit I ran into a few errors. The compiler wasn't happy if where I had placed the wire names since they changed throughout the circuit. After playing around with it though I got it to compile correctly.

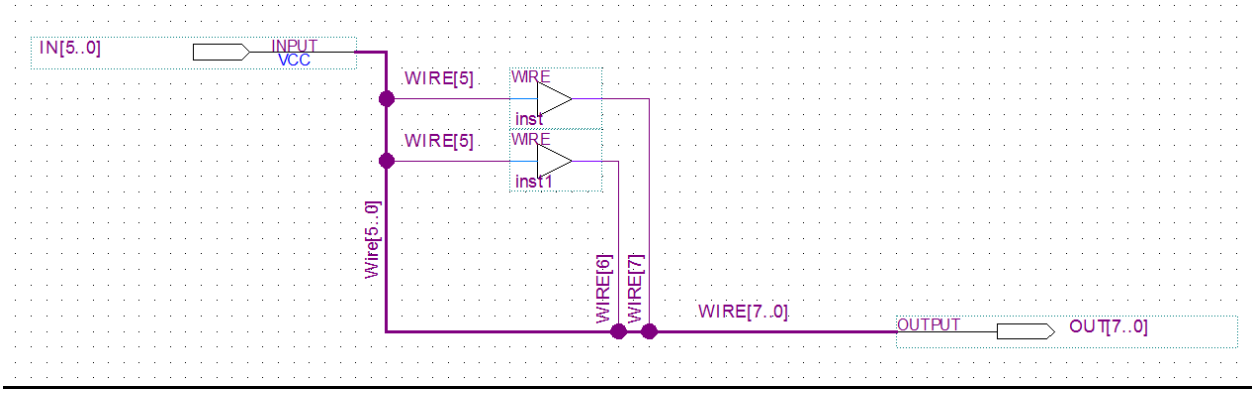
Here is the Block-Diagram for the Zero Fill Circuit.



Part 3 – Extend

This simple circuit is called Extend. Like the previous circuit, this circuit performed a simple but all-important task. It takes a 6-bit input and pads the output with binary values equal to the left most bit in the original input. For example, if the circuit was inputted with 011011 the circuit would output 00011011. Likewise, if the circuit was inputted with 100111 the circuit would output 11100111.

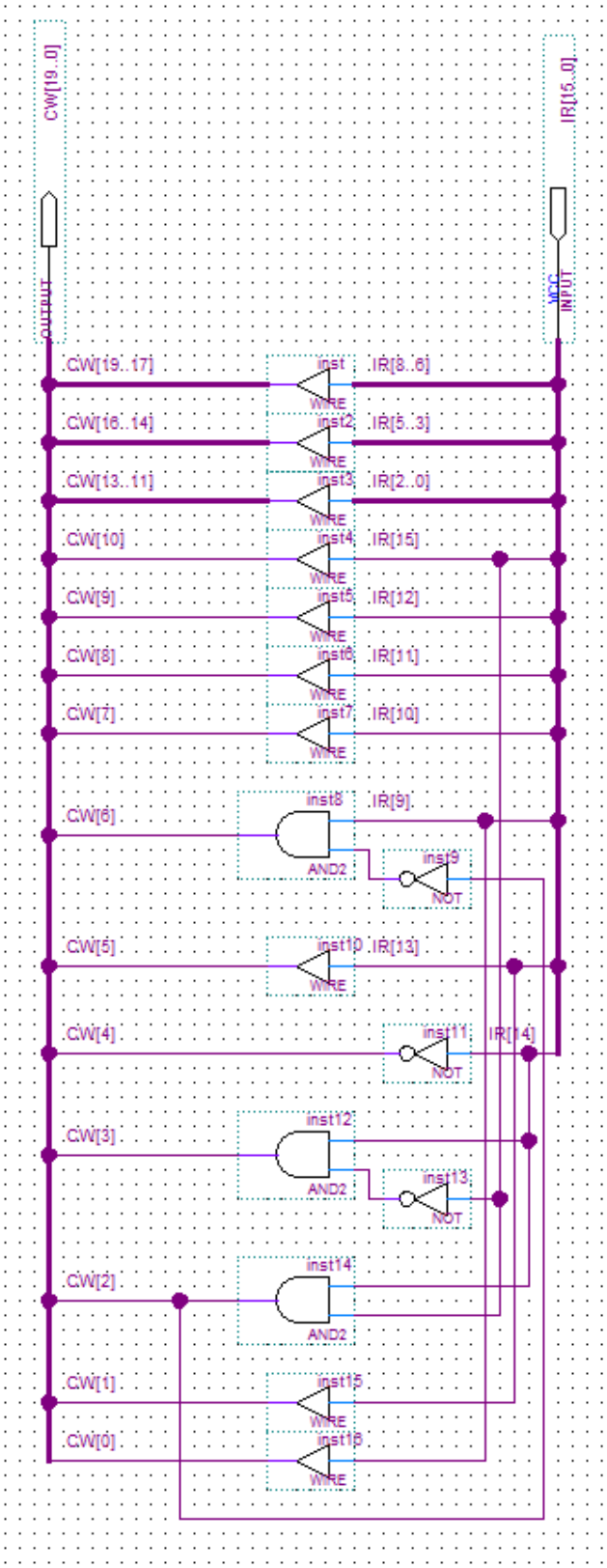
Here is the Block-Diagram for the Extend circuit.



Part 4 – Instruction Decoder

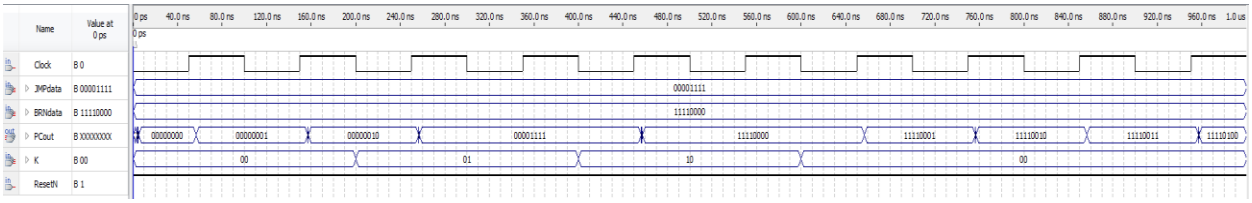
This circuit was called the Instruction Decoder. It performed the task of translating a 16-bit input called the instruction and outputting a usable 20-bit instruction.

Here is the Block-Diagram for the Instruction Decoder.

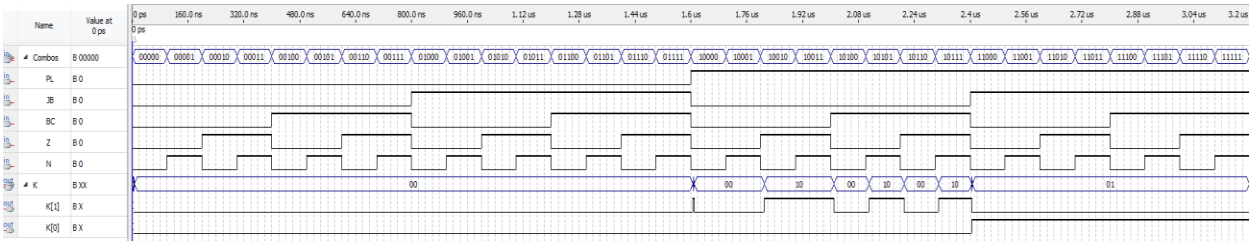


Testing

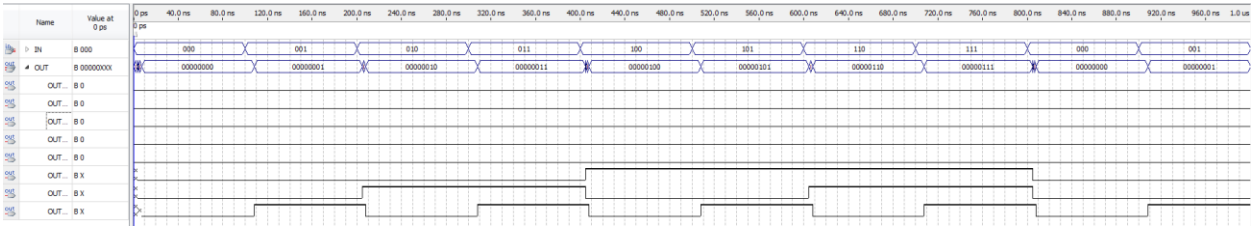
When testing the Program Counter I encountered no problems and it performed as expected for every single input combination.



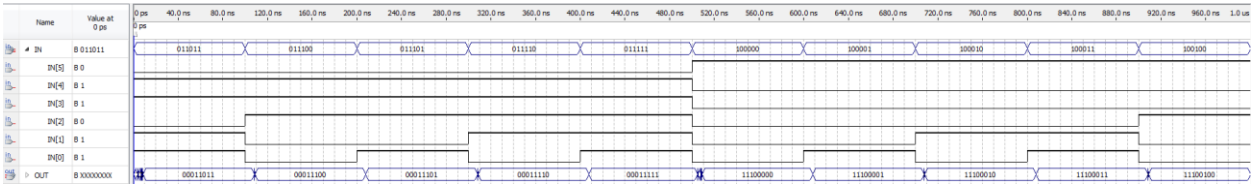
When testing the Branch Control Module I encountered no problems and it performed as expected for every single input combination.



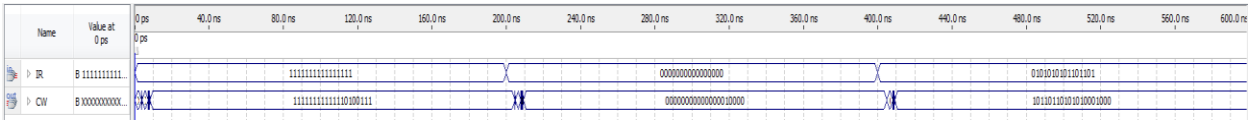
When testing the Zero Fill circuit I encountered no problems and it performed as expected for every single input combination.



When testing the Extend circuit I encountered no problems and it performed as expected for every single input combination.



When testing the Instruction Decoder I encountered no problems and it performed as expected for every single input combination.



Conclusion

In this lab, I learned how to create many of the individual parts contained in our single cycle computer. I learned that to make something as complex as a single cycle computer we need to use many levels of abstraction. For example, many of the components that were created to perform certain tasks contain within themselves boxes that represent other extensive circuits that are reasonably complex in their own right. Without these several layers of abstraction that are used it would be almost impossible to make heads or tails of what the circuit is supposed to do. Overall I am very excited that in the next lab we will assemble all of our individual parts into a hopefully working single cycle computer.