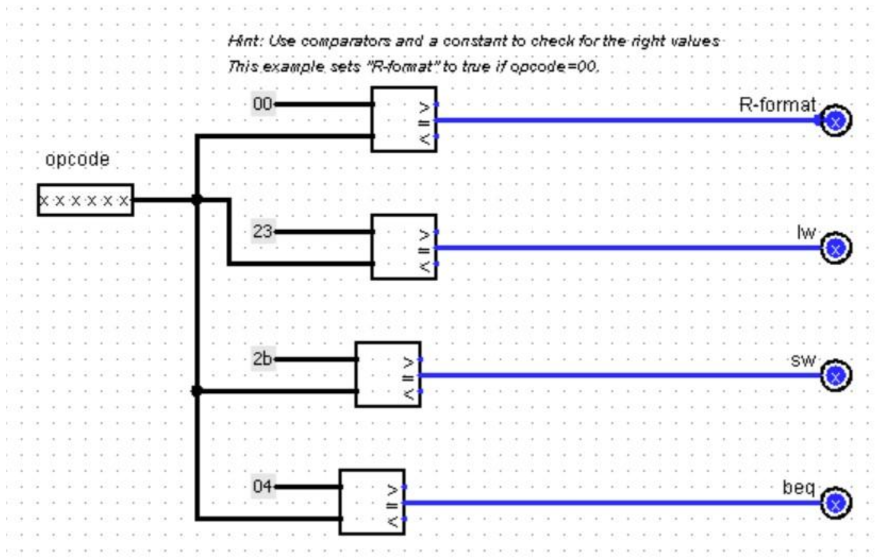
Building a MIPS Processor Lab Report

Mrinisha Adhikari
892309931
June 9, 2023

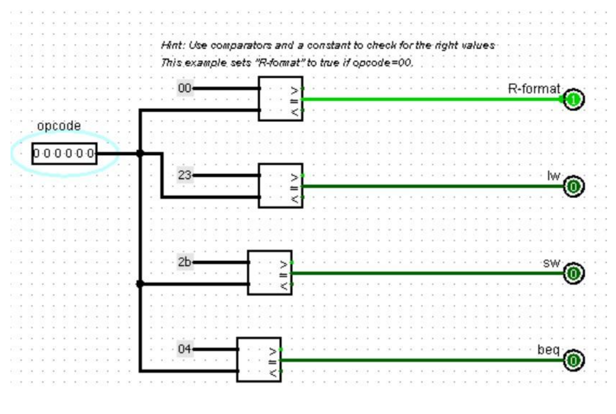
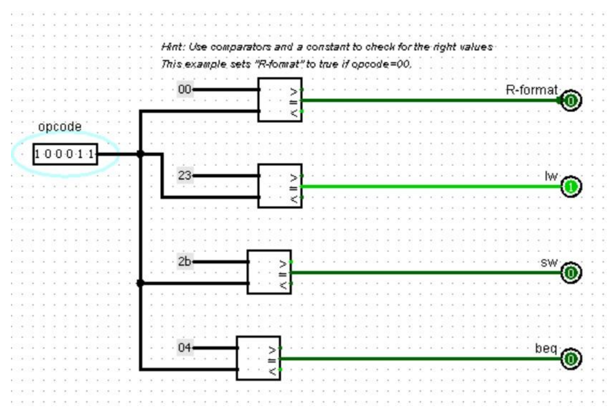
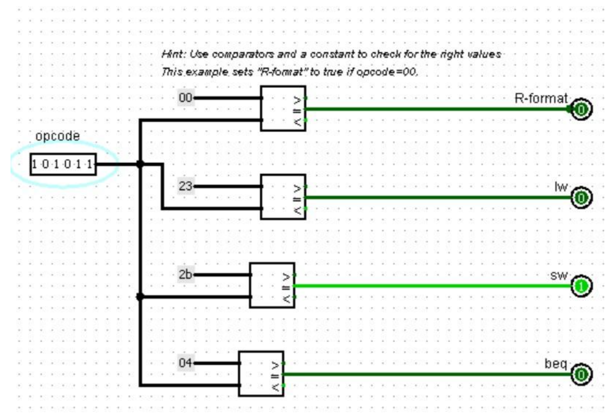
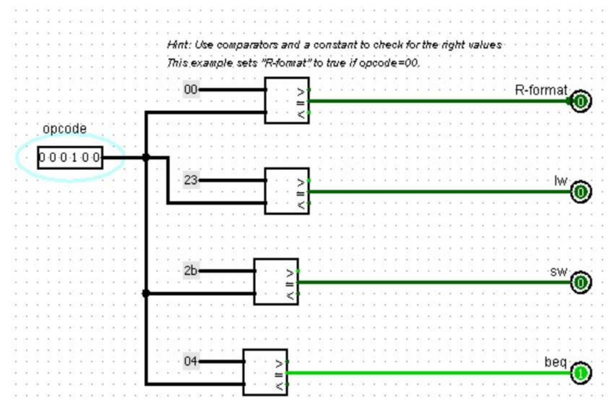
Part 1

ToDo #1:

Complete the Type Decoder Unit by adding comparators to detect the right instruction types.


Hint: Use comparators and a constant to check for the right values
This example sets "R-format" to true if opcode=00.
opcode
xxxxxx
00
23
2b
04
R-format
lw
sw
beq

This is my complete instruction type decoder unit. Each opcode is compared (through the comparator) to its constant hexa-decimal value to decode its correct type. **All test cases:**

<u>ToDo #2:</u>

Complete the given table to specify what the Control Decode Unit should do.

Output	**R-format**	**Lw**	**Sw**	**Beq**
RegDst	1	0	X	X
ALUSrc	0	1	1	0
MemtoReg	0	1	X	X
RegWrite	1	1	0	0
MemRead	0	1	0	0
MemWrite	0	0	1	0
Branch	0	0	0	1
ALUOp1	1	0	0	0
ALUOp0	0	0	0	1

Values indicate the control signals needed with regards to opcode, for decoded instruction type.

ToDo #3:

Finish the Control-Decode circuit. I used the combinational analysis tool to build this.

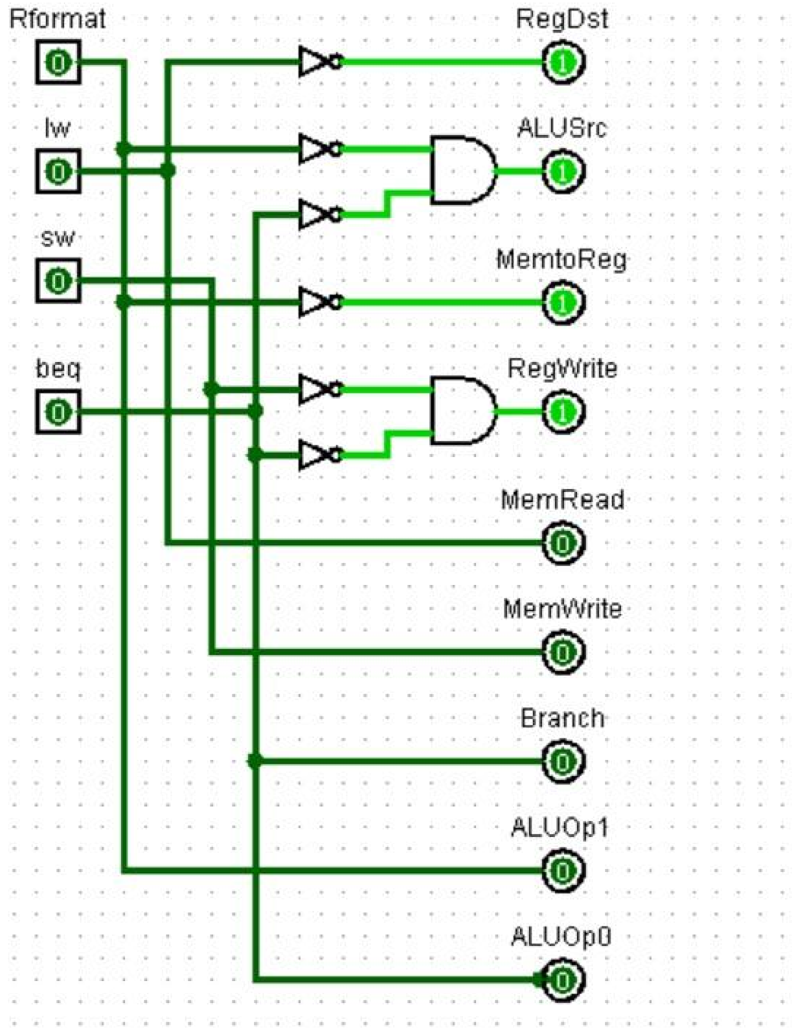
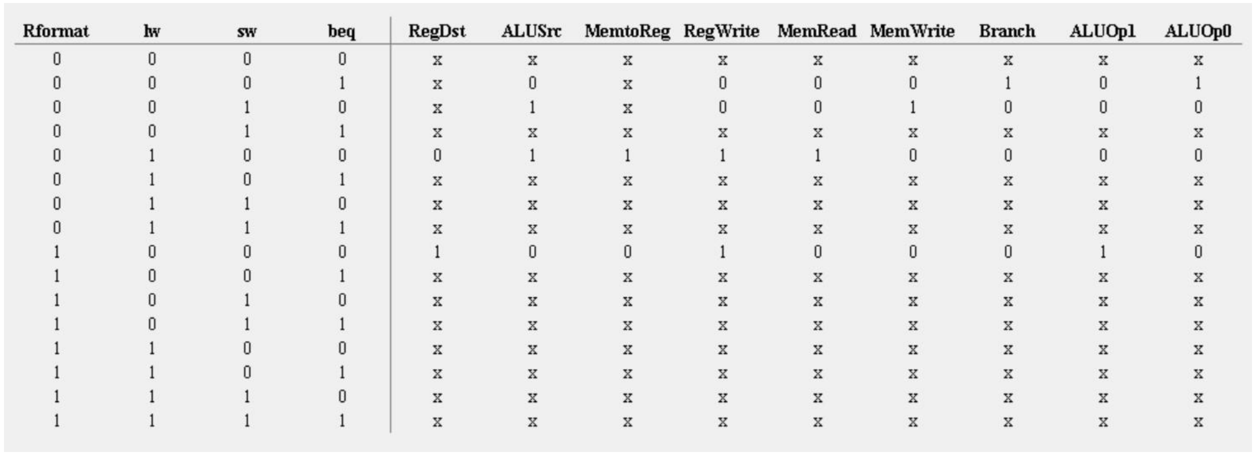
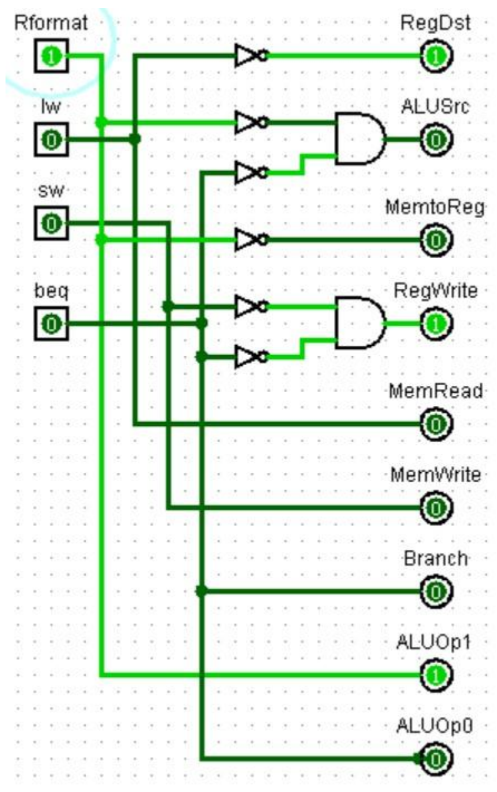


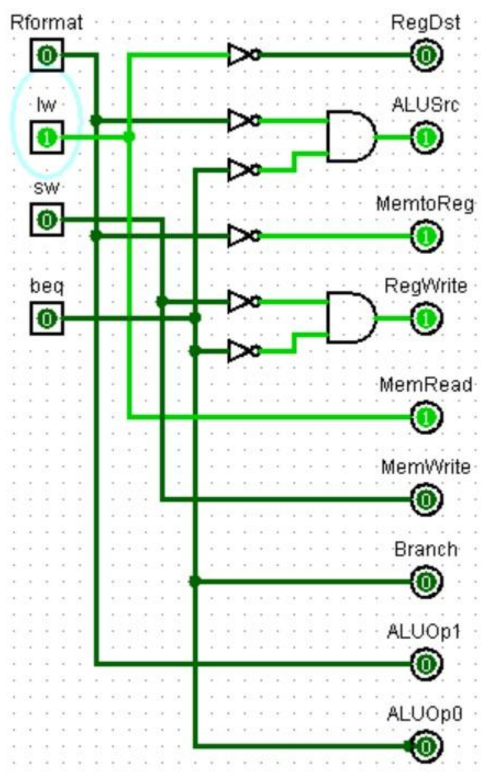
Table i used to built this, for reference:

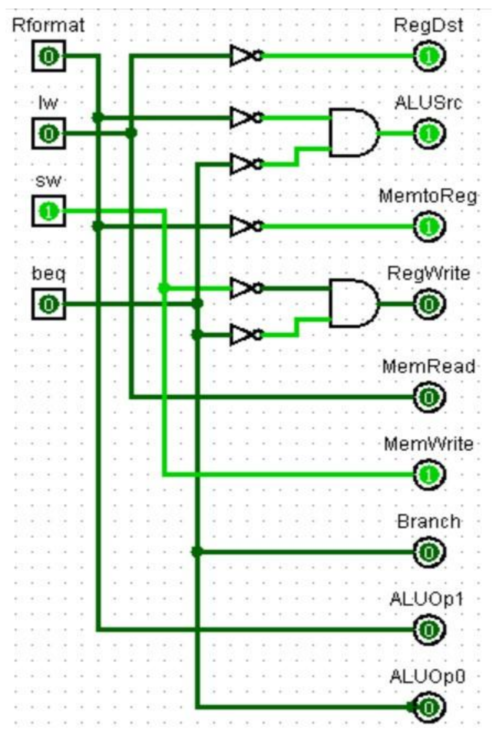


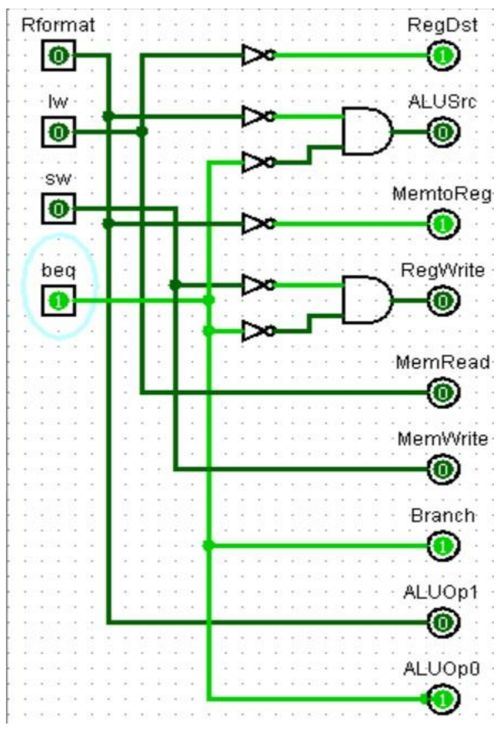
Rformat	lw	sw	beq	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
0	0	0	0	x	x	x	x	x	x	x	x	x
0	0	0	1	x	0	x	0	0	0	1	0	1
0	0	1	0	x	1	x	0	0	1	0	0	0
0	0	1	1	x	x	x	x	x	x	x	x	x
0	1	0	0	0	1	1	1	1	0	0	0	0
0	1	0	1	x	x	x	x	x	x	x	x	x
0	1	1	0	x	x	x	x	x	x	x	x	x
0	1	1	1	x	x	x	x	x	x	x	x	x
1	0	0	0	1	0	0	1	0	0	0	1	0
1	0	0	1	x	x	x	x	x	x	x	x	x
1	0	1	0	x	x	x	x	x	x	x	x	x
1	0	1	1	x	x	x	x	x	x	x	x	x
1	1	0	0	x	x	x	x	x	x	x	x	x
1	1	0	1	x	x	x	x	x	x	x	x	x
1	1	1	0	x	x	x	x	x	x	x	x	x
1	1	1	1	x	x	x	x	x	x	x	x	x

Testing my control-decoder for each instruction type. Comparing values to my table created in to-do#2. *note- values in RegDst in lw and sw don't matter, and values in MemtoReg in sw and beq don't matter.

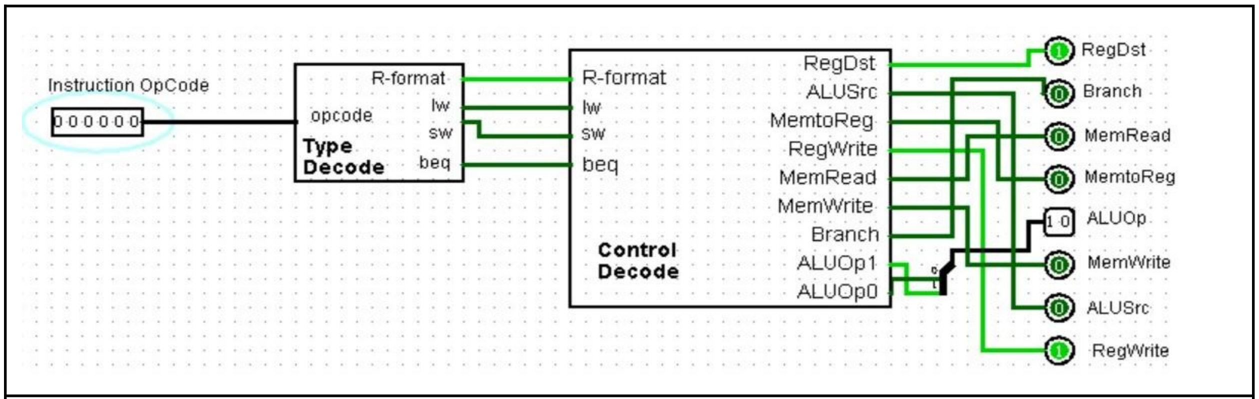


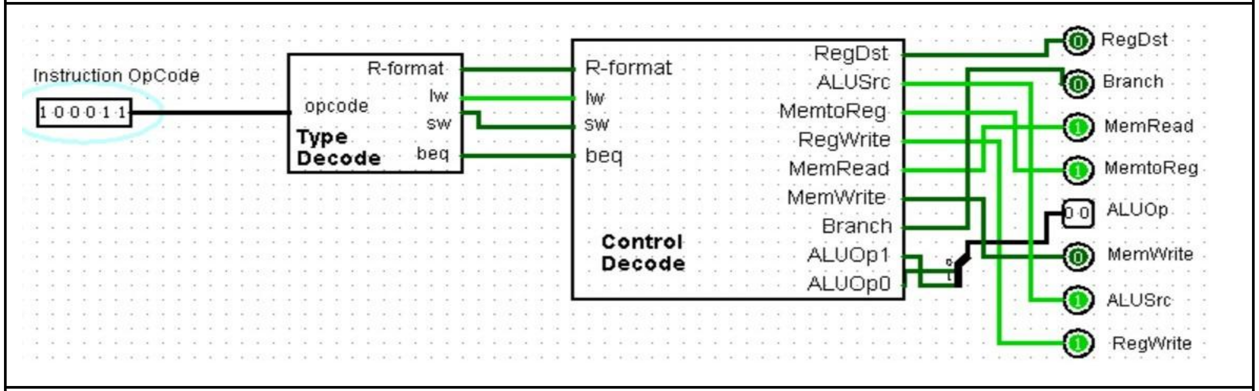


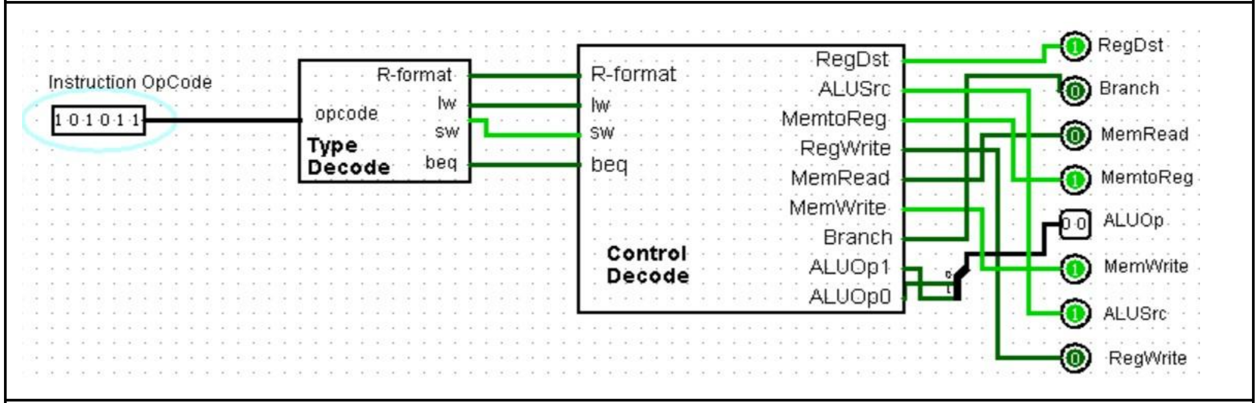


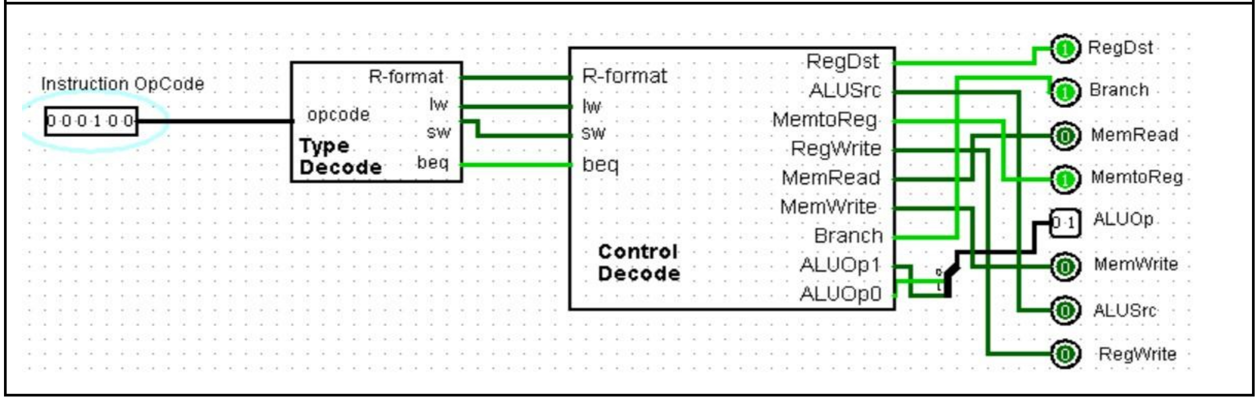


<u>Further verifying my tests in the control unit (per lab instructions):</u>





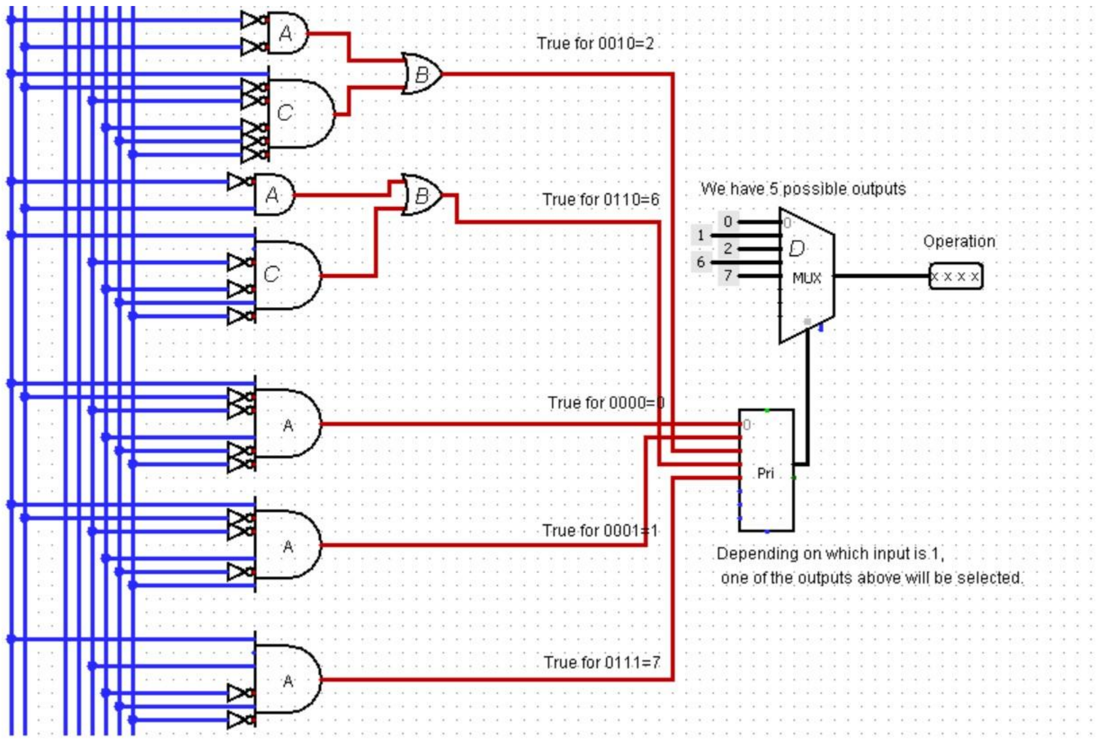




ToDo #4:

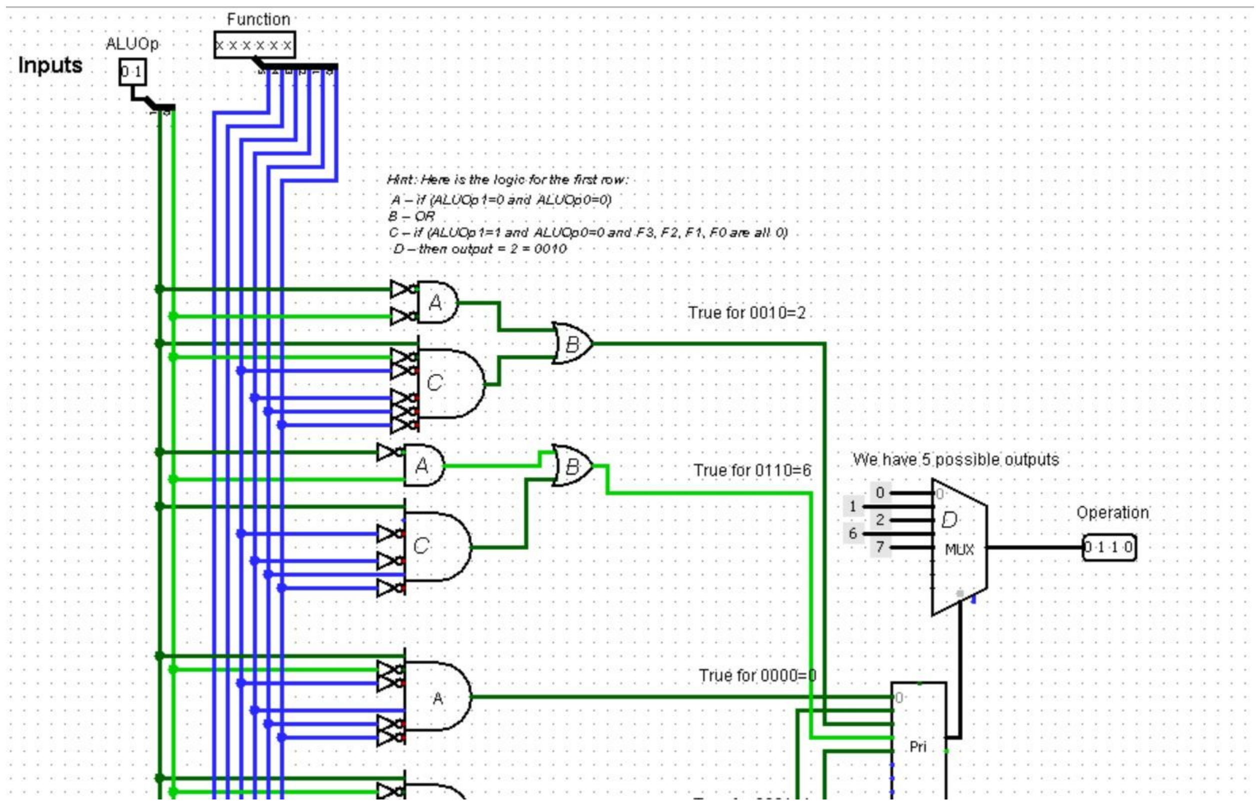
Using the given table, complete the circuit to drive the three other possible outputs.

I derived all four remaining outputs from the table. Complete ALU control unit:



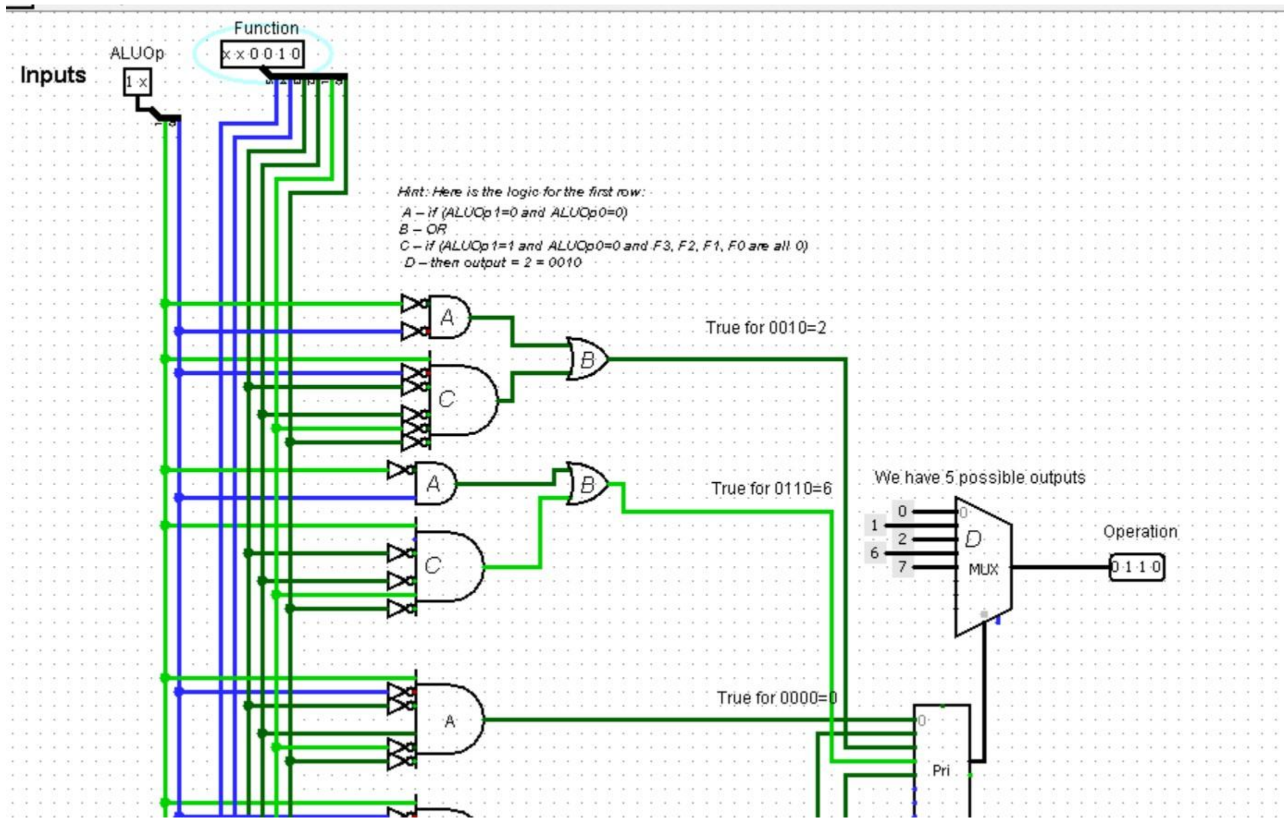
- **First output: 0110**
  A: if (ALUOp1 = 0 and ALUOp0 = 1)
  B: OR
  C: if (ALUOp1 = 1 and ALUOp = x and F3=0,F2=0,F1=1,F0=0)
  D: then output = 6
- **Second output: 0000**
  A: if (ALUOp1 = 1 and ALUOp0 = 0 and F3=0,F2=1,F1=0,F0=0)
  D: then output = 0
- **Third output: 0001**
  A: if (ALUOp1 = 1 and ALUOp0 = 0 and F3=0,F2=1,F1=0,F0=1)
  D: then output = 1
- **Third output: 0111**
  A: if (ALUOp1 = 1 and ALUOp0 = x and F3=1,F2=0,F1=1,F0=0)
  D: then output = 7

Testing my ALU unit, for D= 0110, in both OR cases



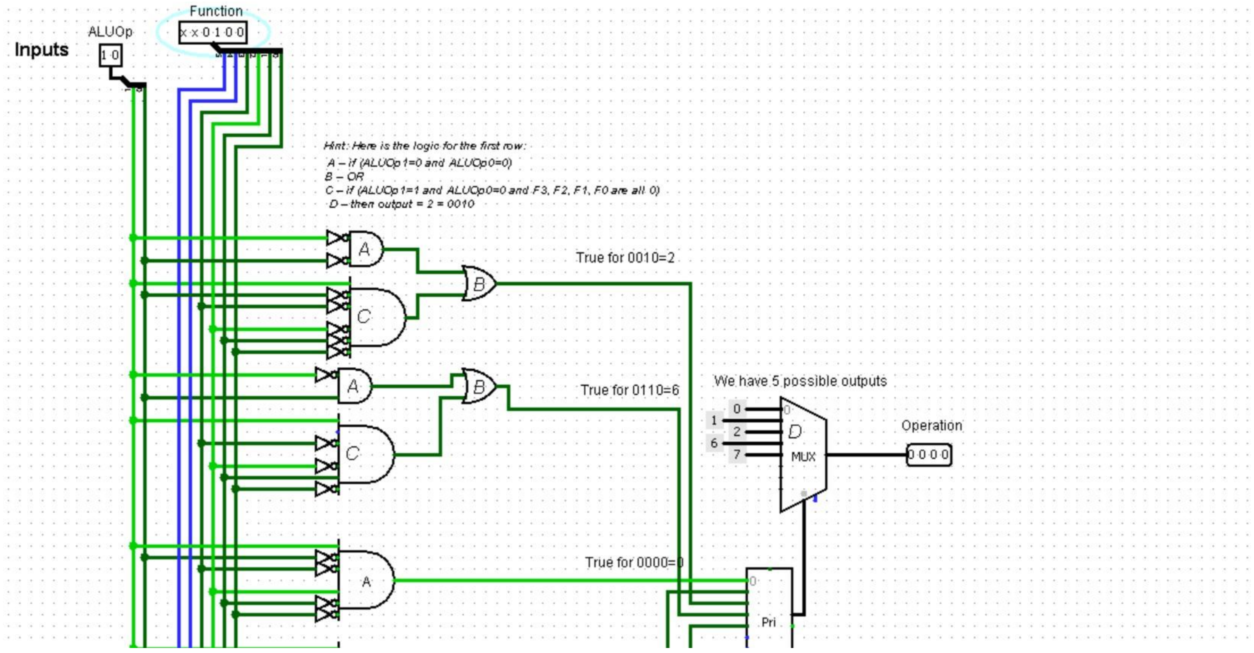
Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010

OR

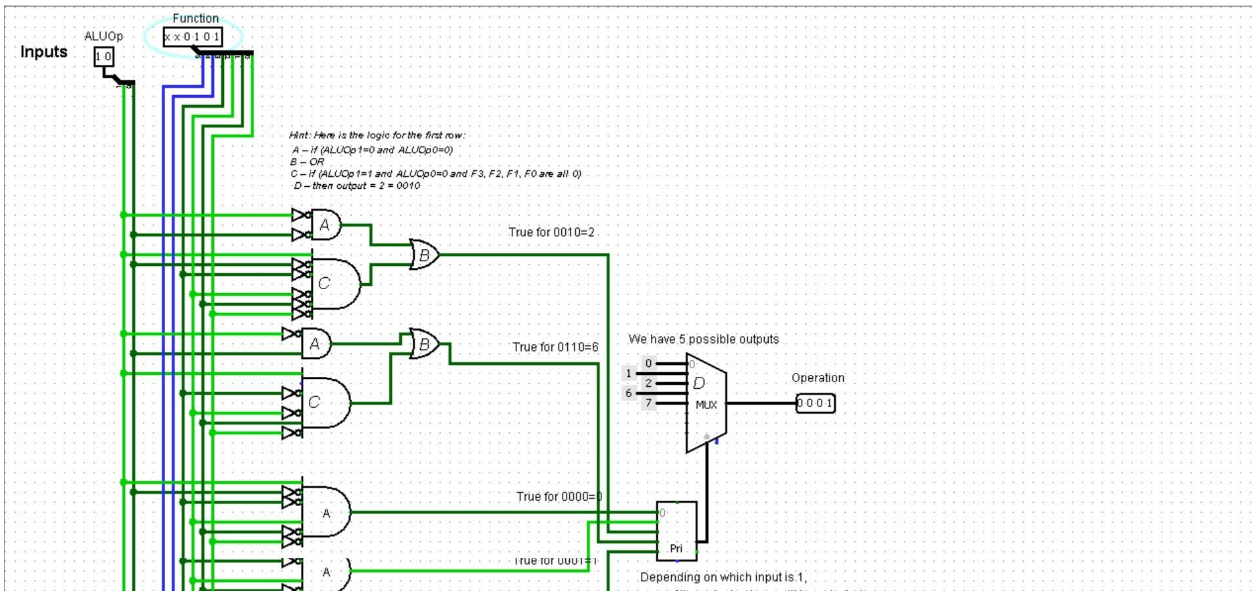


Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010

D = 000 case,

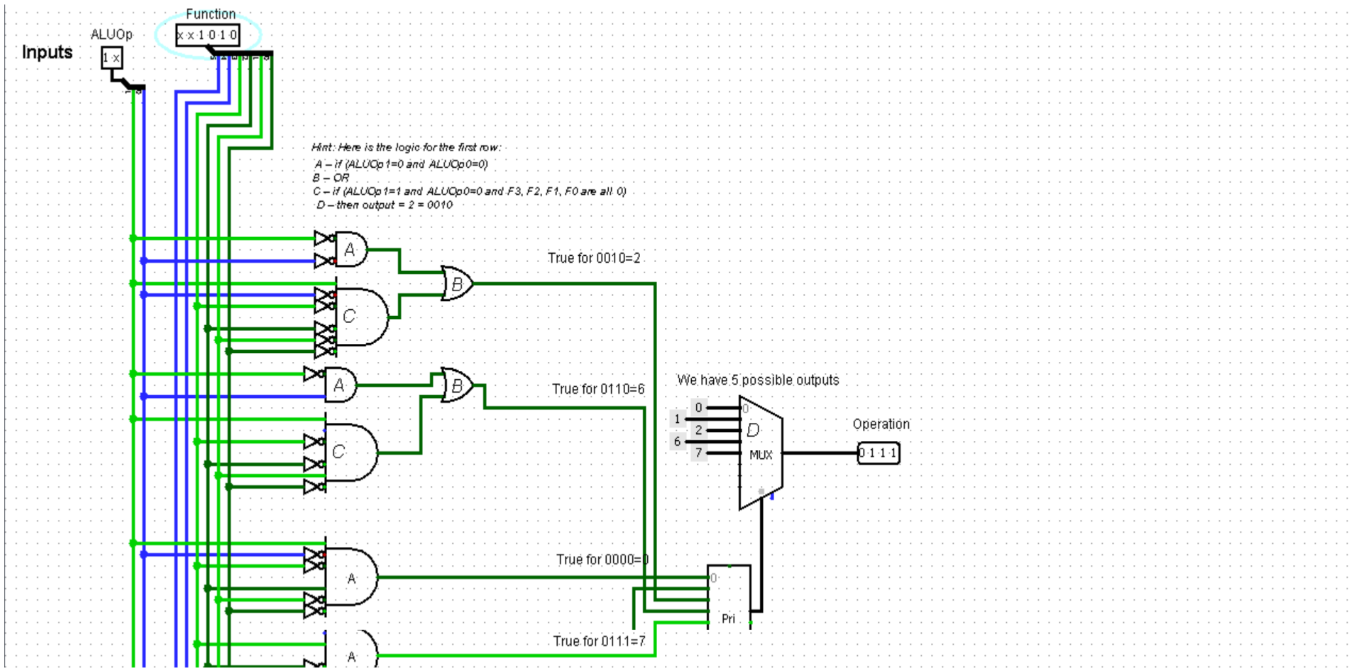

Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010

D = 0001 case,


Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010

Depending on which input is 1,

D = 0111 case,

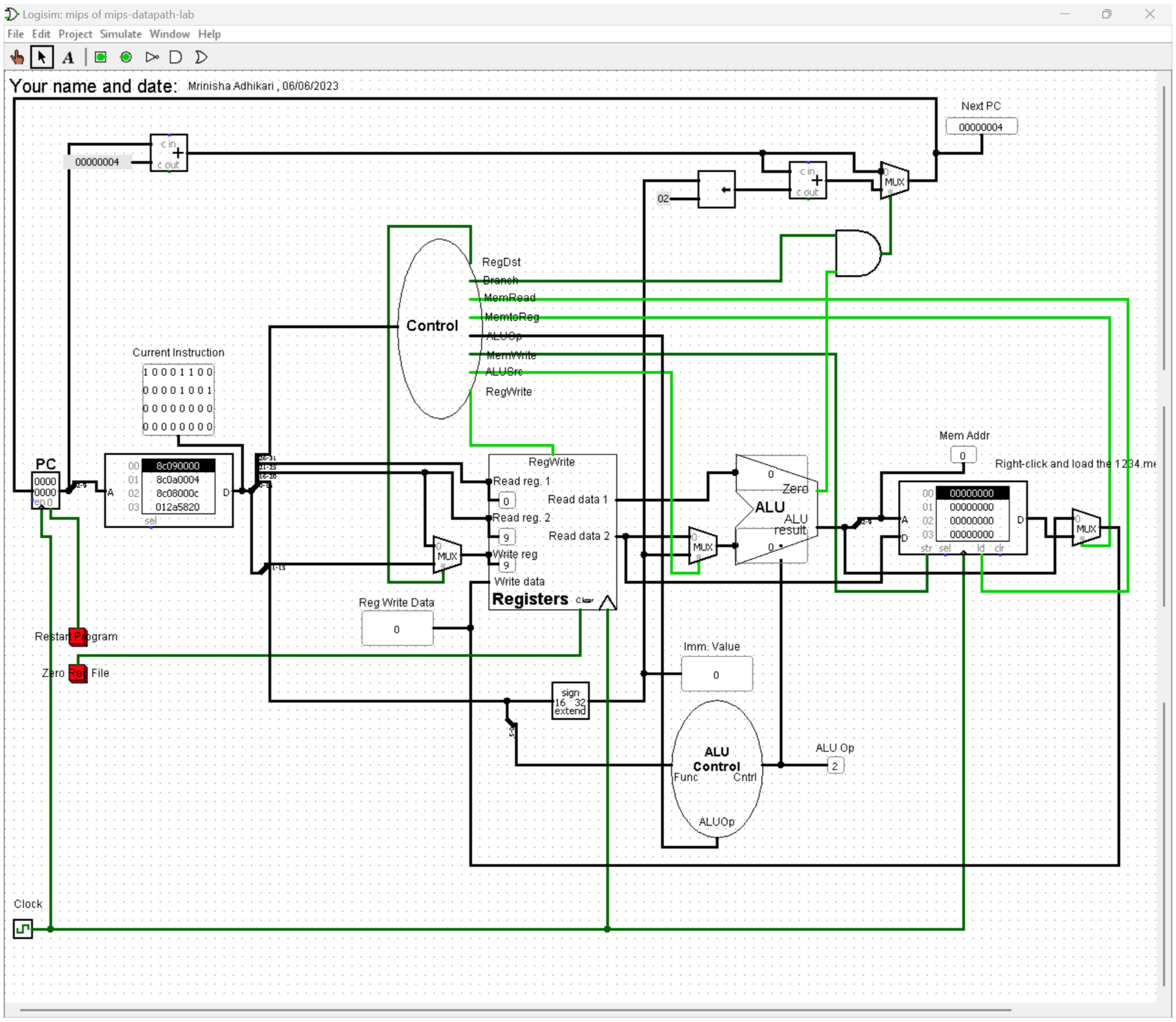


Hint: Here is the logic for the first row:
A – if (ALUOp1=0 and ALUOp0=0)
B – OR
C – if (ALUOp1=1 and ALUOp0=0 and F3, F2, F1, F0 are all 0)
D – then output = 2 = 0010

Part 2

ToDo #5:

Wire up your processor. (Correctly.)

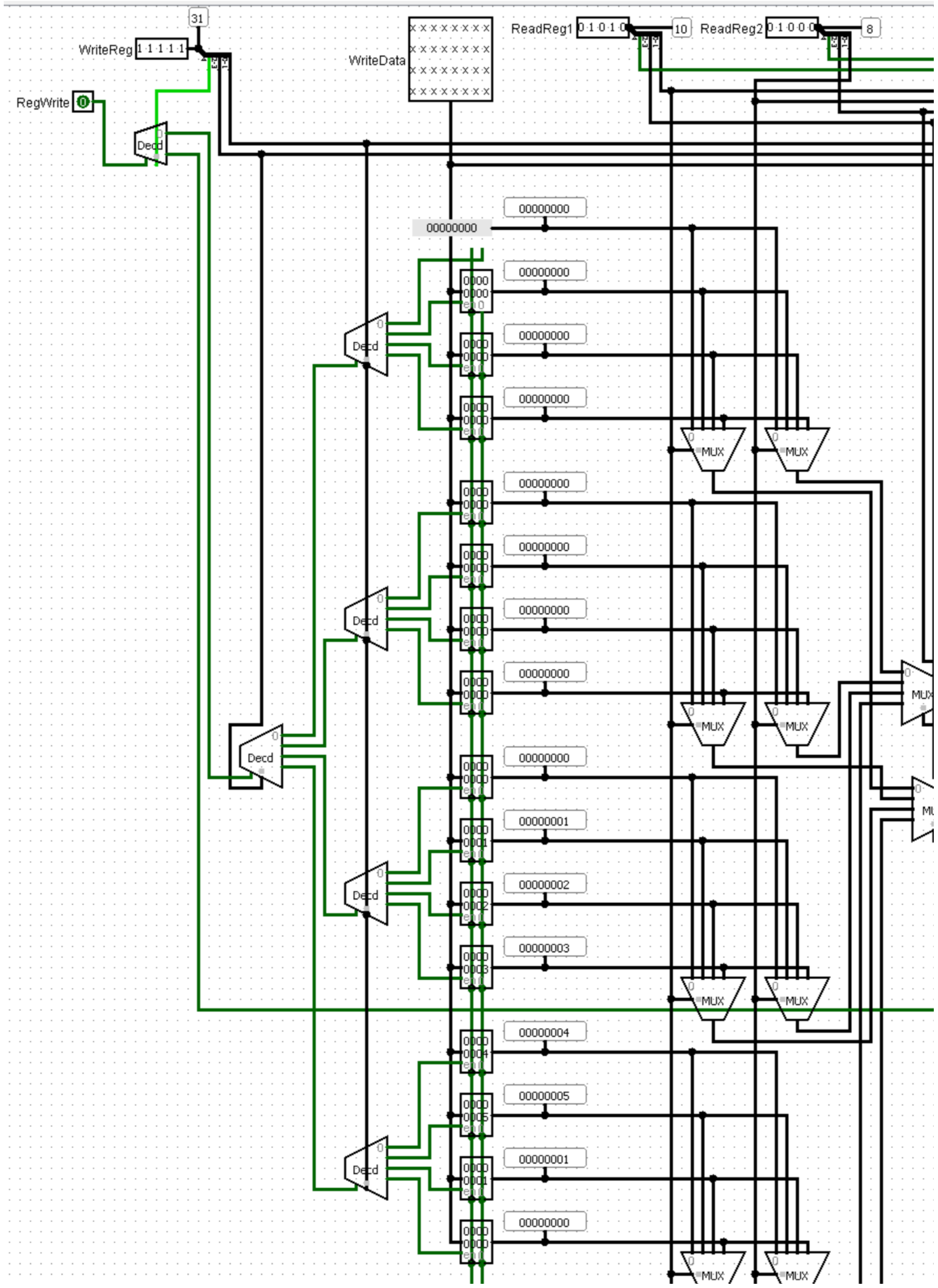


ToDo #6:

What are the values of $t0-$t6 in the register file after executing the program?

- Values of $t0-$t6 should be as so: 0,1,2,3,4,5,1
- Code loops back to 03

Testing, this matches my register file after a full instruction run cycle:



- And my processor loops back to 03 as well

<u>Summary</u>

The lab dealt with various tasks in building a MIPS processor in Logisim. I had to create/complete units such as the control decode, signal tables, ALU. I made sure to test all my parts, after completing each part. The corresponding tests have been added as screenshots. After all units were completed and tested for correct values. I put the whole processor together and ran it using the provided instructions and memory files. The test for this has been added as well.

Logically, the testing code worked as so:

- The first three instructions were set to load words from the memory locations 0,4 and 12 into register $t1, $t2, and $t0 respectively
- Then perform a subtraction operation from $t2 to $to, and store in $t0
- Bitwise OR operation occurs on values in $t4 and $t, and is stored in $t5
- Slt instruction sets $t6 to 1, if $t4 is less than $t5, or to 0 otherwise
- The beq instruction jumps to ‘loop’ if $t2 = $t0, this keeps happening until $t2 != $t0

This logic works with the mem file contents of 1,2,3,4, and gives $t-0$t6 values as: 0,1,2,3,4,5,1. All corresponding workings and values of this are included as screenshots and circ files.

<u>Some debugging that i had to do:</u>

- the SW pin in the control decode seems to have moved in someway after the using the combinational analysis tool

- There seems to be a certain way the outputs in the register show up depending on what simulation option i choose in logisim. Meaning, if i ran the processor through logisim option Simulation->“Ticks enable” , and then manually stopped the program, it would have already taken another cycle into account and the value in $t0 shows up as 2. But if I make sure to ‘restart program’ and ‘zero reg file’, then run the processor through the clock, the output matches perfectly.