Lab 2 :: Logic Lab

Konsing Ham Lopez, 1037, ECS 154A 001

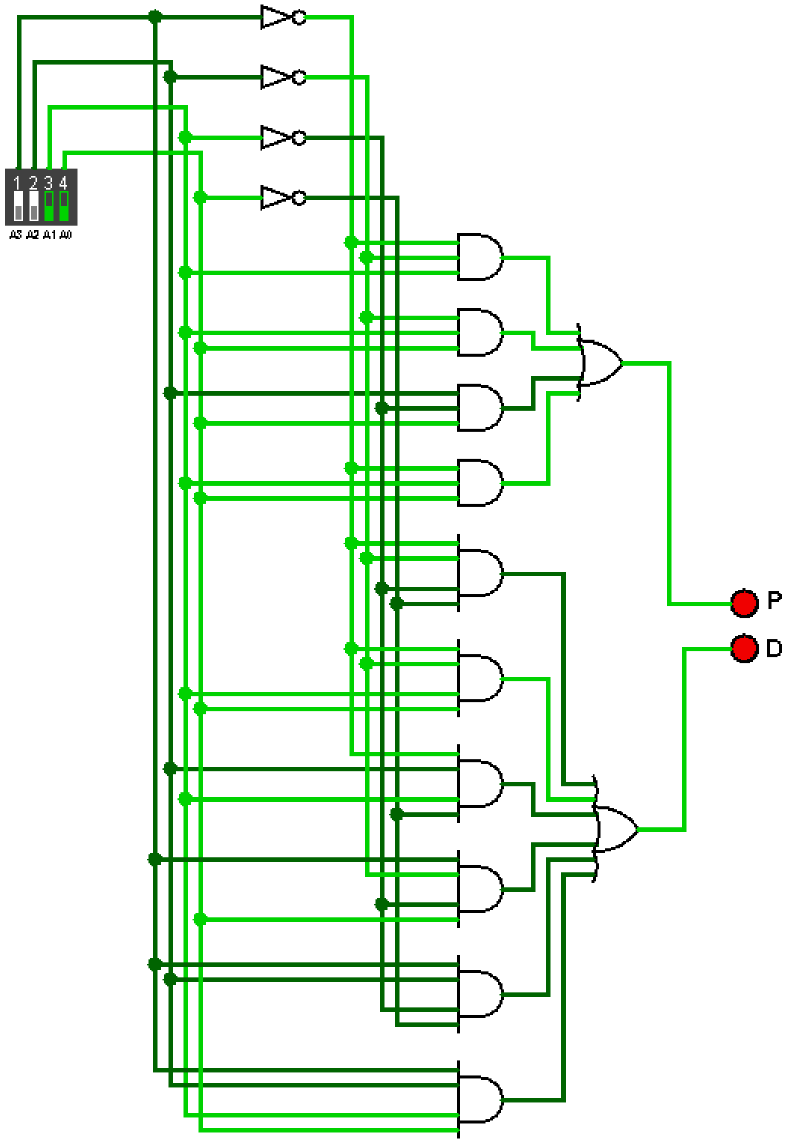
Table of Contents

Introduction

Lab 2 consists of a few problems which I solved using Logisim. Each problem in this document consists of the circuit's schematic, the Karnaugh map, truth table, and theory of operation where applicable.

Problem 1

A3	A2	A1	A0	P	D
0	0	0	0	0	1
0	0	0	1	0	0
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	1	0	1



Karnaugh Map for P.
P = ~A3·~A2·A1 + ~A2·A1·A0 + A2·~A1·A0 + ~A3·A1·A0

A3A2 \ A1A0	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	0	1	0	0
10	0	0	1	0

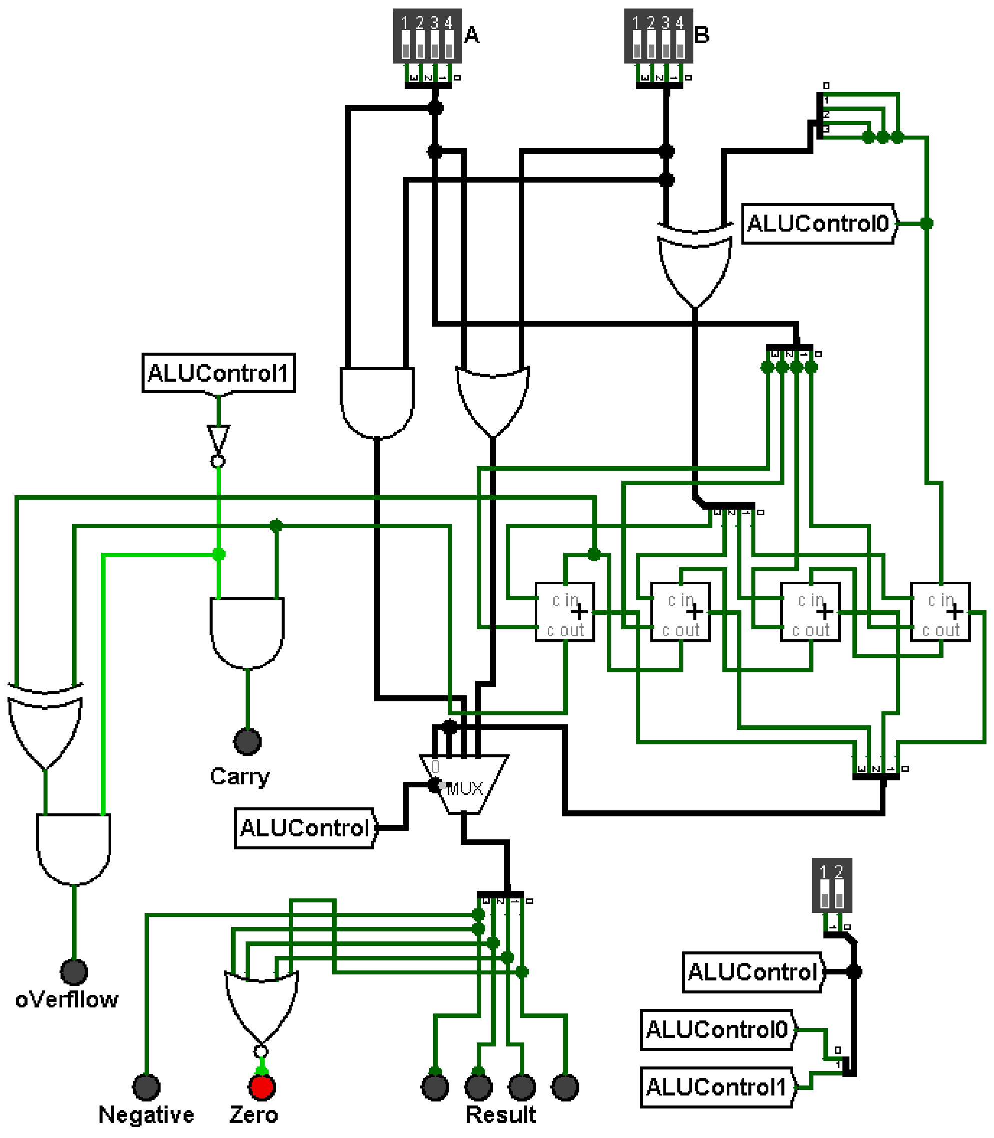
Karnaugh Map for D.
D = ~A3·~A2·~A1·~A0 + ~A3·~A2·A1·A0 + ~A3·A2·A1·~A0 + A3·~A2·~A1·A0 + A3·A2·~A1·~A0 + A3·A2·A1·A0

A3A2 \ A1A0	00	01	11	10
00	1	0	1	0
01	0	0	0	1
11	1	0	1	0
10	0	1	0	0

Theory Of Operation

This circuit contains 2 LEDs, one that indicates if the number is prime and one that indicates if it’s divisible by 3. Included is a truth table that outlines this clearly, alongside 2 karnaugh maps, 1 for each output (P and D). P lights up when the number inputted is prime. D lights up when the number is divisible by 3. The Karnaugh maps are set up so that A3 and A2 are the 2 leftmost bits (shown on the left column) and A1 and A0 are the 2 rightmost bits (shown on top row). The circuit itself has the sum of products of the P output on the top 4 AND gates which all connect to an OR gate. The sum of products of D is represented in the bottom half of the circuit with 6 AND gates which are connected to their corresponding OR gate.

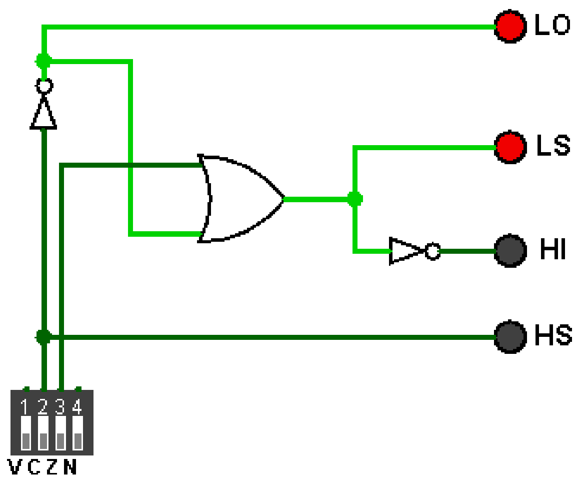
Problem 2



Theory Of Operation

In this circuit, I have 2 tunnels which are connected to a 2 bit dip switch which controls the operation that will be performed. If both bits are off (00), the ALU will perform an add operation (A+B). If only the right bit is on (01), it performs a subtraction (A-B). If both on (11), it performs an OR operation on A and B. If the left bit is on (10), it performs an AND operation on A and B. The rest of the flags are to check for overflow, zero, negative, and carry. The N flag is connected to the most significant bit and checks if the number is negative (that’s why the N flag is connected to the most significant number. The Z flag checks when the entire result is 0 bits and turns on accordingly through the use of a nor gate. The C flag checks if the adder produces a carry out. Lastly, an overflow flag checks if the addition of 2 equal signed numbers produce a result with a different sign (such as negative from positive or vice-versa). This entire process is performed on 2 (4 bit) inputs named A and B. (See Appendix for each of the four operations.

Problem 3



Karnaugh map

```
C Z | LO LS HI HS

-----------------

0 0 | 1  1  0  0

0 1 | 0  0  1  1

1 0 | 1  1  0  0

1 1 | 0  1  0  1
```

No need for Karnaugh maps, equations are:

L = ~Z

LS = C + ~Z

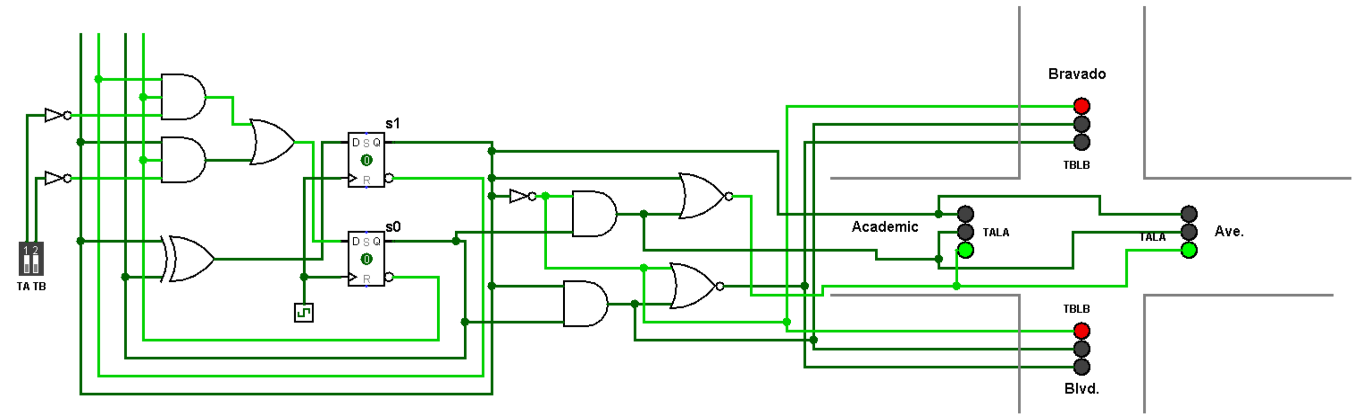
H = ~C·Z

HS = Z

Theory Of Operation

This circuit shows the flags after the subtraction of 2 inputs. The outputs are low, low same, high same, and high which describe the relationship between A and B inputs.The flags used are carry and Zero since the overflow and negative flags are inconsistent.

Problem 4a



State	Encoding
S0	0 0
S1	0 1
S2	1 0
S3	1 1

Light	Encoding
0 0	Green
0 1	Yellow
1 0	Red

Where CS = Current State and NS = Next State

CS		Input		NS	
S1	S0	Ta	Tb	S’1	S’0
0	0	0	x	0	1
0	0	1	x	0	0
0	1	x	x	1	0
1	0	x	0	1	1
1	0	x	1	1	0
1	1	x	x	0	0

CS		Output Generator			
S1	S0	LA1	LA0	LB1	LB0
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Using the output generator above, LA1, LA0, LB1, LB0 are:

LA1 = S1

LA0 = ~S1·S0
LB1 = ~S1
LB0 = S1·S0

Karnaugh map for S’0.
S’0 = ~S1·~S0·~TA + S1·~S0·~TB

S1S0|TATB	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

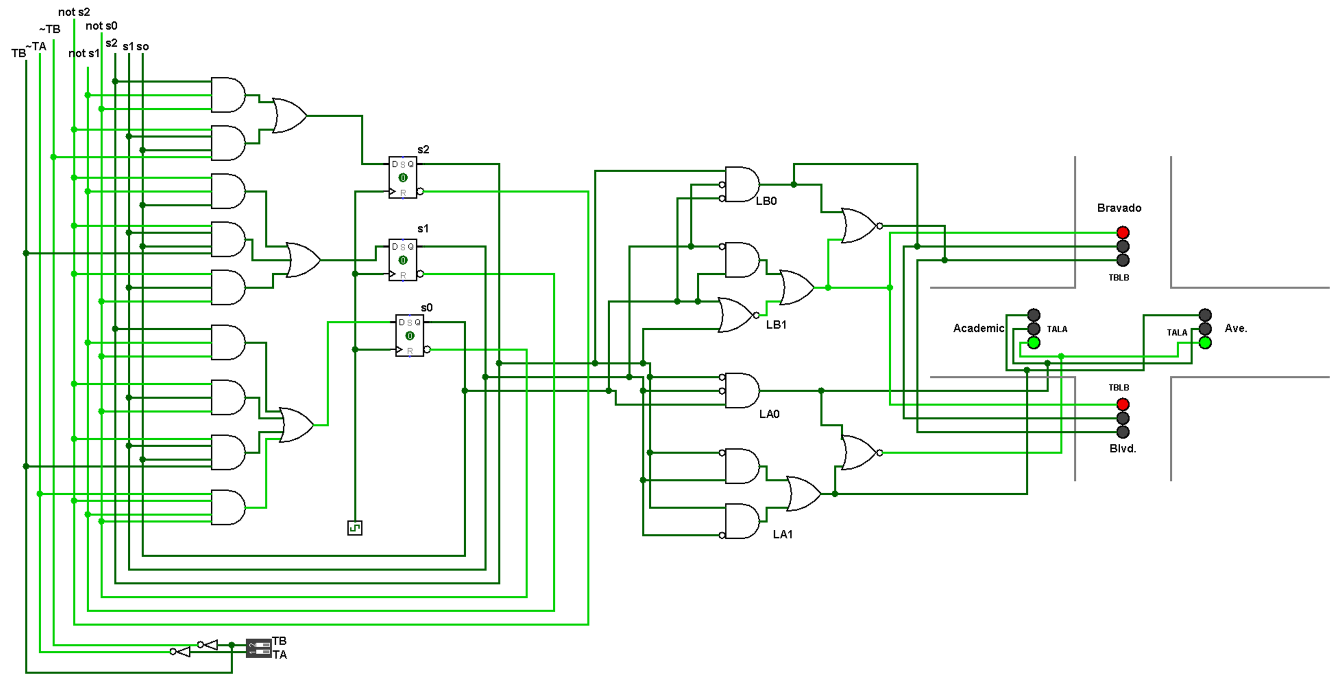
Karnaugh map for S’1
S’1 = S1 ⊕ S0

S1S0|TATB	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

Theory Of Operation

For this problem, the green light is encoded as 00, the yellow light as 01, and the red light as 10. Furthermore, state S0 is encoded as being 00, S1 being 01, S2 being 10, and S3 being 11. The state transition table (which helps create the next state generator) and output generator tables have also both been provided. The output generator was formed with the help of the FSM from class. The next state generator is also formed thanks to the FSM. In addition, the Karnaugh maps for S’1 and S’0 have been added. The circuit itself consists of the TA/TB inputs on the far left, the next state generator on the left, the 2 d flip flops that compute the next state in the middle, the output generator on the right side which handles the lights, and the traffic lights themselves on the right side. The lights go from red green, to red yellow, to green red, to yellow red, and loops back to red green.

Problem 4b



State	Encoding
S0	0 0 0
S1	0 0 1
S2	0 1 0
S3	0 1 1
S4	1 0 0
S5	1 0 1

Light	Encoding
Green	0 0
Yellow	0 1
Red	1 0

Where CS = Current State and NS = Next State

CS			Input		NS		
S2	S1	S0	Ta	Tb	S’2	S’1	S’0
0	0	0	0	x	0	0	1
0	0	0	1	x	0	0	0
0	0	1	x	x	0	1	0

CS			Outputs			
S2	S1	S0	La1	La0	Lb1	Lb0
0	0	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	1	0	1	0

```
0  1  0  | x  x  |  0   1   1          0  1  1  |  1   0   0   0
0  1  1  | x  0  |  1   0   0          1  0  0  |  1   0   0   1
0  1  1  | x  1  |  0   1   1          1  0  1  |  1   0   1   0
1  0  0  | x  x  |  1   0   1
1  0  1  | x  x  |  0   0   0
```

Karnaugh Map for S’0
S’0 = ~S2·~S1·~S0·~TA + ~S2·S1·S0·TB + ~S2·S1·~S0 + S2·~S1·~S0

TATB|S2 S1 S0	000	001	011	010	110	111	101	100
00	1	0	0	1	0	0	0	1
01	1	0	1	1	0	0	0	1
11	0	0	1	1	0	0	0	1
10	0	0	0	1	0	0	0	1

Karnaugh Map for S’1
S’1 = ~S2·~S1·S0 + ~S2·S1·S0·TB + ~S2·S1·~S0

TATB|S2 S1 S0	000	001	011	010	110	111	101	100
00	0	1	0	1	0	0	0	0
01	0	1	1	1	0	0	0	0
11	0	1	1	1	0	0	0	0
10	0	1	0	1	0	0	0	0

Karnaugh Map for S’2
S’2 = ~S2·S1·S0·~TB + S2·~S1·~S0

TATB|S2 S1 S0	000	001	011	010	110	111	101	100
00	0	0	1	0	0	0	0	1

01	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	1
10	0	0	1	0	0	0	0	1

Karnaugh Map for LA1
LA1 = S2·~S1 + ~S2·S1

S2|S1S0	00	01	11	10
0	0	0	1	1
1	1	1	0	0

Karnaugh Map for LA0
LA0 = ~S2·~S1·S0

S2|S1S0	00	01	11	10
0	0	1	0	0
1	0	0	0	0

Karnaugh Map for LB1
LB1 = ~S2·~S0 + ~S1·S0

S2|S1S0	00	01	11	10
0	1	1	0	1
1	0	1	0	0

Karnaugh Map for LB0
LB0 = S2·~S1·~S0

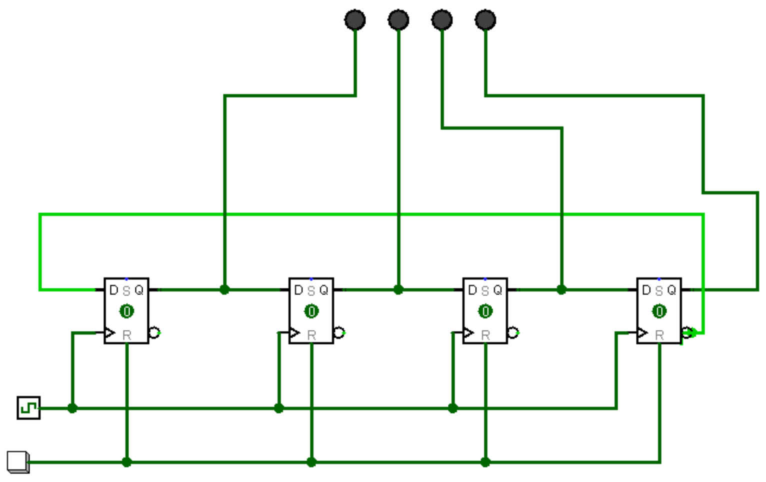
S2|S2S0	00	01	11	10

0	0	0	o	0
1	1	0	o	0

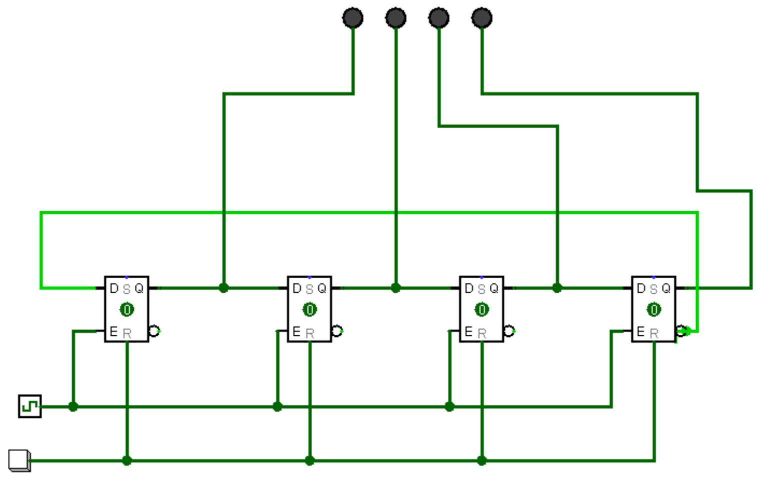
Theory Of Operation

For this problem, the green light is encoded as 00, the yellow light as 01, and the red light as 10. Furthermore, state S0 is encoded as being 000, S1 being 001, S2 being 010, S3 being 011, S4 being 100, and S5 being 101. The state transition table (which helps create the next state generator) and output generator tables have also both been provided. The output generator was formed with the help of the FSM from class, with the addition of 2 extra (red-red) states after S1 and S4. The next state generator is also formed thanks to the FSM. In addition, the Karnaugh maps for S’2, S’1 and S’0 have been added. The circuit itself consists of the TA/TB inputs on the bottom, the next state generator on the left, the 3 d flip flops that compute the next state in the middle left, the output generator on the middle right (which handles the lights), and the traffic lights themselves on the right side. The lights go from red green, to red yellow, to red red, to green red, to yellow red, to red red, and loops back to red green. The Karnaugh maps for LB1 provides the gates for red light, LB0 for yellow light, LA1 for the other red light, LA0 for the other yellow light, and the NOR of the yellow and red lights (for each street) provides the green light wire.

Problem 5a Schematic



Part B Schematic (D flip flops set to High level)



a) The number sequence would be 0000 -> 1000 -> 1100 -> 1110 -> 1111 -> 0111 -> 0011 -> 0001-> back to beginning
b) The johnson counter would take a total of 8 cycles if we're dealing with 4 bits (so 2*n cycles where n is the number of bits). In other words, each arrow represents a cycle. The reason 2*N represents the total cycles because you first run N cycles (fill all bits with 1s through right shifting) then go back to all 0s by shifting to the right with 0s again.
d) The Johnson counter benefits from simplicity and efficiency. For example, the Johnson counter has the same number of flip flops a ring counter would have, but is able to handle twice as many states. Additionally, the Johnson counter is a self-decoding circuit meaning that it can depend on itself when creating following inputs. For example, the Johnson counter's latches take in 2 inputs and make a unique input from them. This is more efficient and reliable than others such as the binary counter which requires new inputs before every output.

Theory Of Operation

PartA

The Johnson counter works by shifting a 1 to the right until all bits are filled with a 1, then shifting to the right and filling with 0 bit each time. This circuit essentially uses the data from previous cycles and keeps looping. The circuit itself consists of 4 D flip flops and connects the Q output to the D input of the next D flip flop (except the last one which connects the ~Q output to the first D flip flop's D input. They each have a clock input and a reset button that resets the counter back to 0. Considering it’s a 4 bit counter, it has a total of 8 cycles.

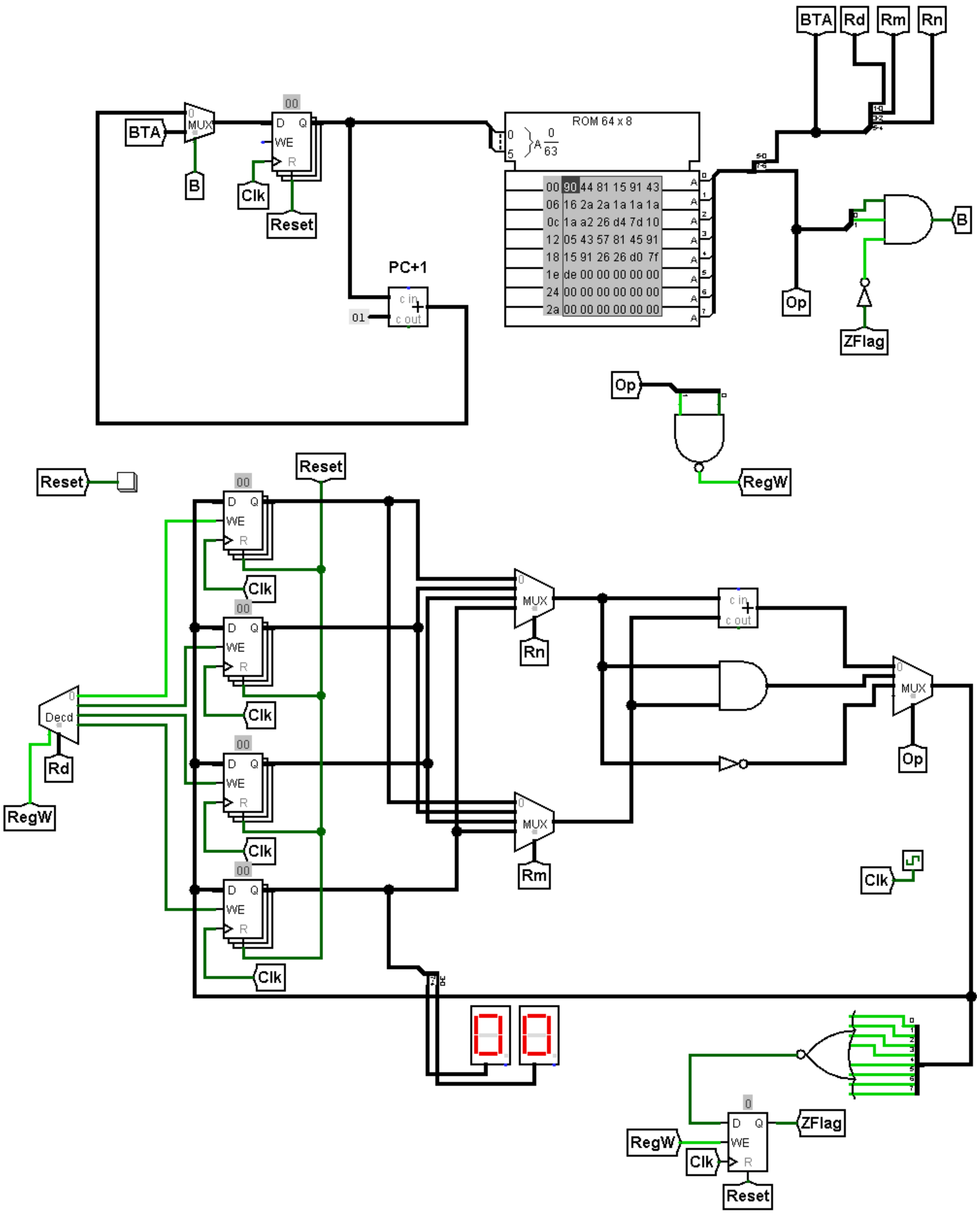
PartB

The Johnson counter with d-latches (D flip flops set to High level) runs into the signal chasing problem. In simpler terms, this is an issue that occurs when using d latches rather than flip flops. In this scenario, the inputs shoot straight through all the d latches instead of going stopping at the inverter on the clock (which a D flip flop would have). In other words, the input goes straight to the output. In Logisim, this error is detected as an “oscillating error” that stops the program shortly after runnin

Problem 6

Truth table for controller

Operation	Z	RegW	B
00	0	1	0
00	1	1	0
01	0	1	0
01	1	1	0
10	0	1	0
10	1	1	0
11	0	0	1
11	1	0	0



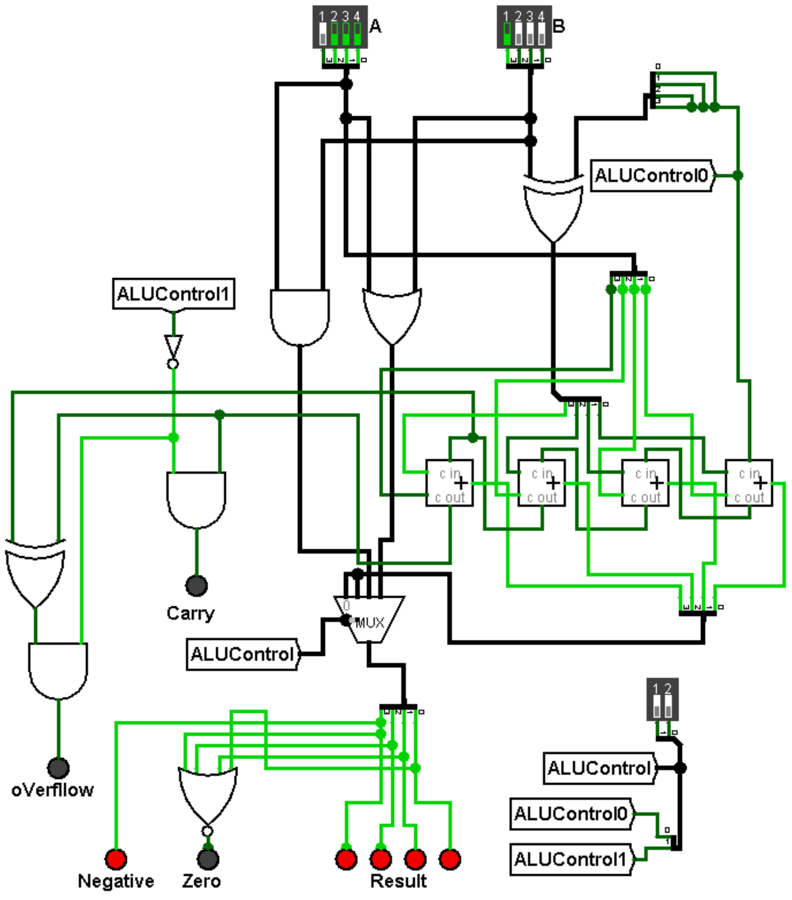
```
00 90 44 81 15 91 43
06 16 2a 2a 1a 1a 1a
0c 1a a2 26 d4 7d 10
12 05 43 57 81 45 91
18 15 91 26 26 d0 7f
1e de 00 00 00 00 00
24 00 00 00 00 00 00
2a 00 00 00 00 00 00
```

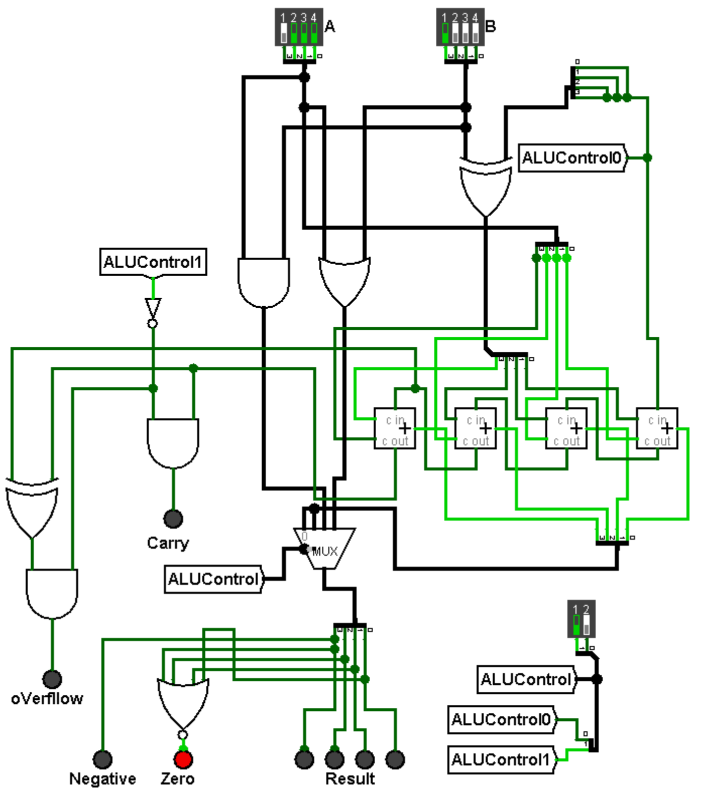
Theory Of Operation

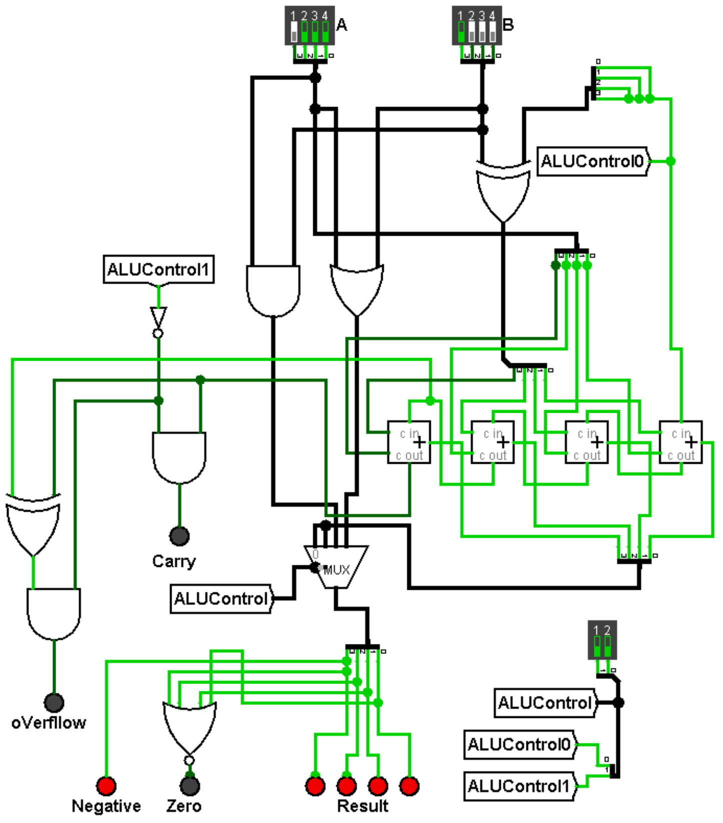
Problem 6 is essentially building the fiscsim that we created in Lab 1. The circuit itself is very similar to the one shown in the discussion a few weeks ago and works by taking a .h file (hex) such as fibo2.h which is passed into the ROM (which contains 64 8 bit addresses). The program counter register to the left of the ROM takes in a reset and clock, and the 1 bit adder increases the program counter (starting from the 0th position). This is how the circuit knows which instruction to fetch. Furthermore, a mux on the left side of that assists with BNZ operations that provide a branch target address which, in turn, could change the program counter to a specific address. On the right side of the ROM, the OP code (operation code such as ADD, AND, and NOT is checked). The data bits from the ROM are also sent through the BTA, Rd, Rm, and Rn registers accordingly to be used (depending on the RegW). They're also sent to an AND gate with the inversion of Z flag to B which checks if you branch or not. You branch only if b is not zero. The op code is also passed into an AND gate which checks if we should enable register write. The circuit should only write if the instruction is not a BNZ. The bottom section of the schematic (from left to right) includes a decoder which takes in the display register (Rd) and the register write tunnel (RegW), and determines which register Rd should write to. It then passes the information to the corresponding registers in the register file and these registers then pass the information into 2 muxes which are connected to the register files (and determine which registers the data should be read from) and are ultimately passed into the AND, ADD, and NOT gates to perform the adequate operation. This data is then passed into a mux which is connected to the op code. So for example, if the operation is ADD (00) the mux on the right side would add the Rn and Rm bits. The process is exactly the same for NOT and AND. This data is then looped back into the register files and also sent to a NOR gate and register file that sets our Z flag depending on whether it's a write instruction or not. This is why the RegW is being passed in as well. In simpler terms, the Z flag is set to 1 if the previous instruction placed in the display register (Rd) was 0. I also added a display to show what would be in the R3 register such as in the FISCSIM from Lab 1.

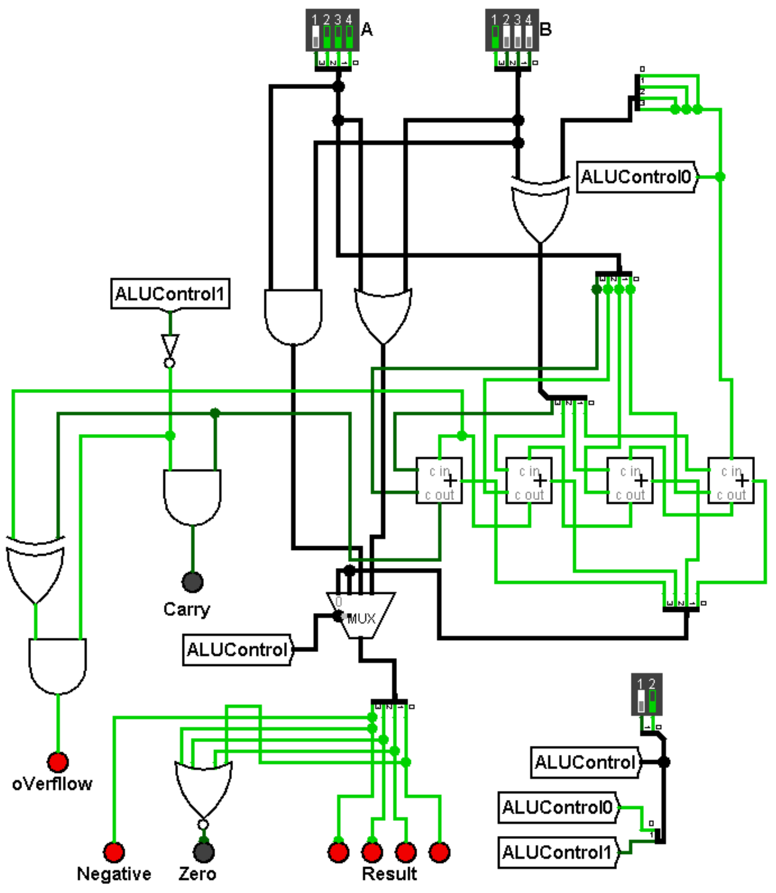
Appendix

For problem 2, four states:




A
B
ALUControl0
ALUControl1
Carry
ALUControl
oVerflow
Negative
Zero
Result
ALUControl
ALUControl0
ALUControl1


A
B
ALUControl0
ALUControl1
Carry
ALUControl
oVerflow
Negative
Zero
Result
ALUControl
ALUControl0
ALUControl1


A
B
ALUControl0
ALUControl1
Carry
oVerfllow
ALUControl
MUX
Negative
Zero
Result
ALUControl
ALUControl0
ALUControl1