ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 1

DUE DATE : 19.05.2021

GROUP NO : G23

GROUP MEMBERS:

150180010 : ELİF ARIKAN

150180112 : ÖMER MALİK KALEMBAŞI

150180052 : HÜSEYİN AVERBEK

1 INTRODUCTION

In this project, we designed and implemented 2 different types of registers, a register file, an Arithmetic Logic Unit the organization which is given in the project.

2 PROJECT PARTS

This project is divided into 4 main sections, Part 1 and Part 2, which are divided into two subsections, Part 3 and Part 4

2.1 PART 1

In this part, we implemented 2 different types of registers: 8-bit register and 16-bit register

2.1.1 PART 1A

As shown in Figure 1, there are following components in our design:

- Multiplexer: We used 4:1 multiplexer. Input FunSel connected to multiplexer and selects which operation going to implement according to function table in assignment file.
- 8-bit Adder: : We used 8-bit full adder to implement the increment operation in function table. First input of the adder connected to OUTPUT, other is connected to 8-bit constant value 1. Output of the adder is connected to multiplexer.
- 8-Bit Subtractor: We used 8-bit subtractor to implement the decrement operation in function table. First input of the subtractor is connected to OUTPUT, other is connected to 8-bit constant value 1. Output of the subtractor is connected to multiplexer.
- D Flip-Flop: We used eight D flip-flops. Each of them stores 1-bit of the data. They all connected an input ENABLE. When ENABLE is high, flip-flops are activated.

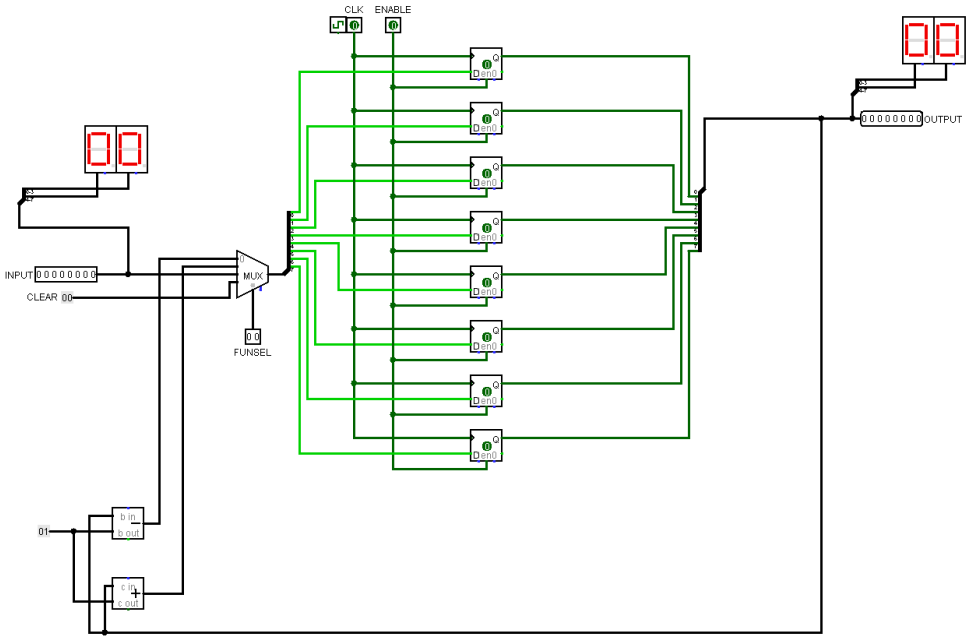

Figure 1: Logisim Circuit For Part-1a

2.1.2 PART 1B

As shown in Figure 2, there are following components in our design:

- Multiplexer(2:1): We used 2:1 multiplexer. Input L/H connected to multiplexer. When L/H is low, INPUT going to be loaded in the last 8-bits(8-15) of the 16-bit multiplexer input. When it is high, INPUT going to be loaded in the first 8-bits(0-7) of the 16-bit multiplexer input.

- Multiplexer(4:1): We used 4:1 multiplexer. Input FunSel connected to multiplexer and selects which operation going to implement according to characteristic table in assignment file.

- 16-bit Adder: : We used 16-bit full adder to implement the increment operation in characteristic table. First input of the adder is connected to OUTPUT, other is connected to 16-bit constant value 1. Output of the subtractor is connected to 4:1 multiplexer.

- 16-Bit Subtractor: We used 16-bit subtractor to implement the decrement operation in characteristic table. First input of the subtractor is connected to OUTPUT, other is connected to 16-bit constant value 1. Output of the subtractor is connected to 4:1 multiplexer.

- D Flip-Flop: We used sixteen D flip-flops. Each of them stores 1-bit of the data.

They all connected an input ENABLE. When ENABLE is high, flip-flops are activated.

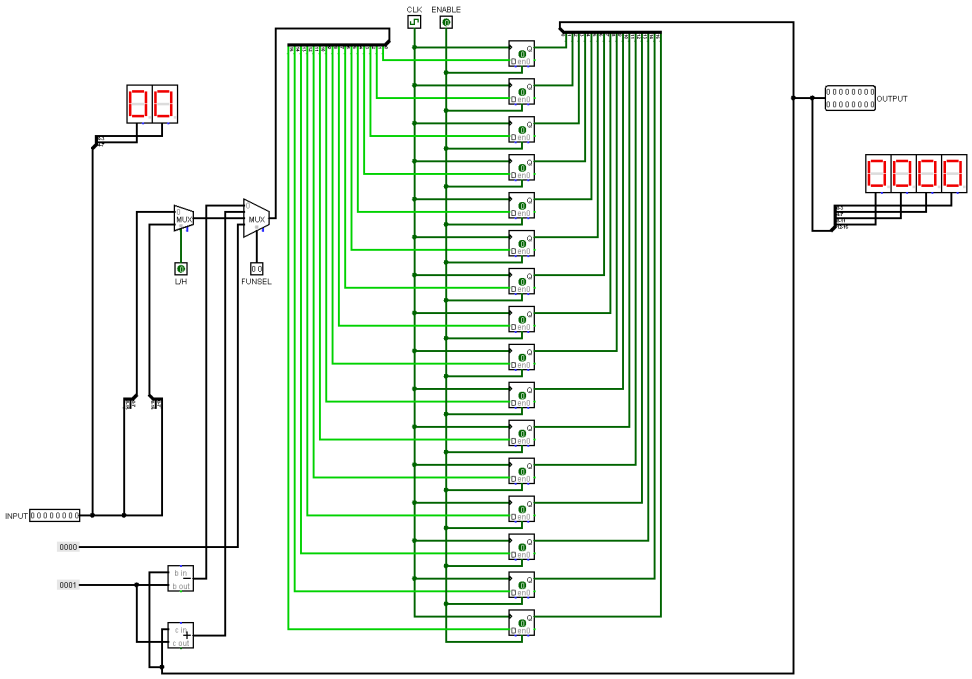

Figure 2: Logisim Circuit For Part-1b

2.2 PART 2

2.2.1 PART 2A

As shown in Figure 3, there are following components in our design:

- 8-bit Register: We used eight 8-bit registers which we designed earlier for Part1a. 4 of these registers are general-purpose registers, R1, R2, R3 and R4. Other 4 are temporary registers, T1, T2, T3 and T4. Input RegSel selects which general-purpose registers going to be enabled and input TmpSel selects which temporary registers going to be enabled.

- Multiplexer: We used two 16:1 multiplexers. Input OutASel connected to first multiplexer and selects which register output going to become output OutA. Input OutBSel connected to second multiplexer and selects which register output going to become output OutB.

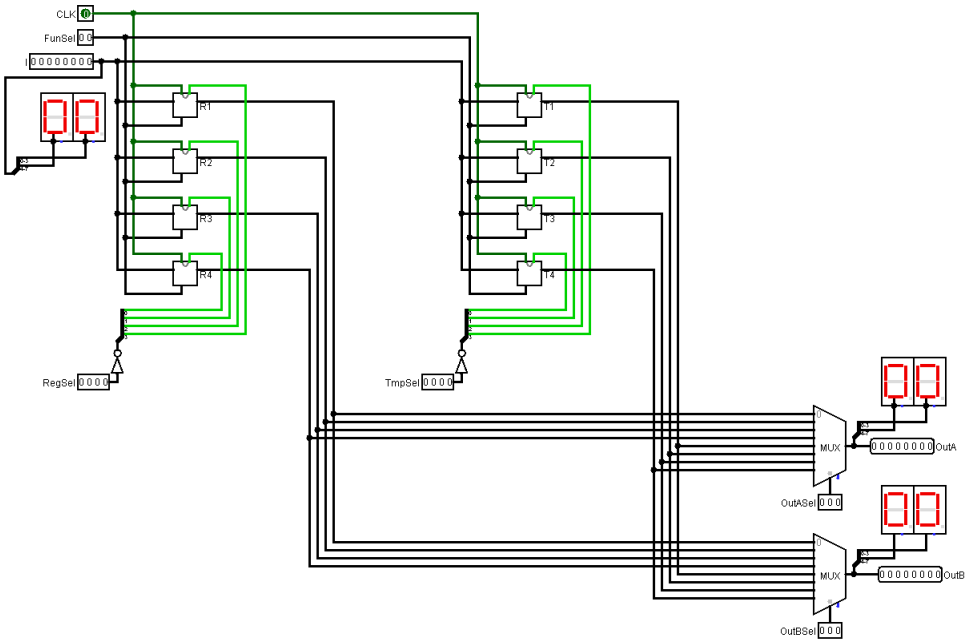

Figure 3: Logisim Circuit For Part-2a

2.2.2 PART 2B

As shown in Figure 4, there are following components in our design:

- 8-bit Register: We used three 8-bit registers which we designed earlier for Part1a. These are Program Counter(PC), Address Register(AR) and Stack Pointer. Input RegSel selects which registers going to be enabled.
- Multiplexer: We used two 4:1 multiplexers. Input OutCSel connected to first multiplexer and selects which register output going to become output OutC. Input OutDSel connected to second multiplexer and selects which register output going to become OutD.

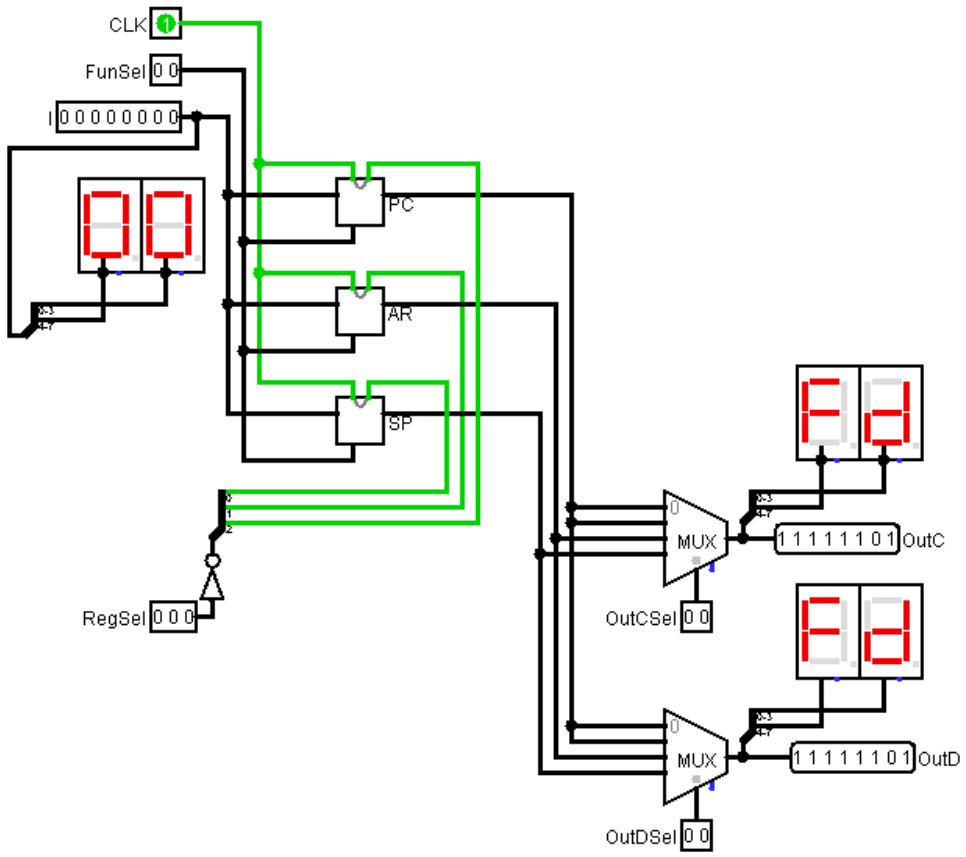

Figure 4: Logisim Circuit For Part-2b

2.3 PART 3

As shown in Figure 11, there are following components in our design:

- Multiplexer: We used a 4:16 MUX to map our inputs through different functions to OutALU. We have 16 functions in total so we used 4-bit FunSel selection input.
- FunSel: 0000 to 0011 This part is made of straightforward operations, A, B, A^{-}, B^{-}. For A and B, we simply connected two lines to each of the inputs. For complements, we used Logisim's inverter gate.
- FunSel: 0100 to 0110 - Arithmetic Operations We implemented three arithmetic operations, in a different subcircuit called 'arithmetic-op'. We use a full adder inside, and we use the 0th and 1st bits of FunSel to use this full adder for adding, adding with carry, subtraction using 2's complement. For adding (FunSel=0100), Cin of full adder in $arithmetic_{op}$ is 0 . For adding with carry (FunSel=0101), we supplied carry input to full adder in arithmetic-op. For subtraction(FunSel=0110), we complemented B and gave 1 to carry input of full adder so it became 2's complement. In addition, if the signs of the operands are the same, the resulting sign must be the

same, otherwise, we can say that there is an overflow. Also, we can view subtraction as addition, using 2's complement. Using this information, we developed a circuit using XOR gates to detect overflow.

- FunSel : 0111 to 1001 - In this part, we connect A and B to AND, OR and XOR gates and connect their output to our MUX respectively.

- FunSel : 1010 and 1011 - Logic Shifts We divided the input signals into 8 and used the 0th bit of our FunSel to select the right or left shift operation. We connected the 8 divided bits of our inputs to the 1X2 mux according to the rule of the shift operation and we used the 0th bit of the FunSel as select input. We fed static 0 to the empty inputs of the MUX. We used a common selection line and output 8 bits.

- Funsel: 1100 and 1101 - Arithmetic Shifts There is no carry in arithmetic shifts, unlike logical shift. Shifting left is adding 0 to the right, which we managed by giving each signal to the MUX one below. Shifting right is copying sign bit, so we fed each signal to the MUX one above and fed A7 to the last MUX again. Also, there is overflow in ASL, so we checked for it by using XOR on the most significant of input and output.

- FunSel: 1110 and 1111 - Circular Shifts When shifting to the right in circular shift operation, the A0th bit of the input is removed as carryout and the value received as carryin is replaced by the A7 bit. When shifting to the left, A7 th bit is removed as carry out and when the value taken as carryin is shifted to the left, it replaces the idle A0 bit. The value received as carryin comes from the zcno flag register of alu.

- ALU Flag Designs There are four flags we keep track of in our design. Z, C, N and O. Z is zero flag, we simply OR all output bits and invert it to decide Z flag. C is carry flag. Carry is updated in arithmetic operations, LSL, LSR, CSR and CSL operations. All circuits that do these operations output a carry. We feed all carry outputs to a MUX, and this MUX decides which operation is being done, so it outputs that operations carry flag. This output goes to a controlled buffer, which has enable input. We use a decoder and OR gate to select cases where carry is updated, and feed this to enable input. N is negative flag, we decide it using the MSB of the output and XOR'ing it with 0. O is overflow flag. Overflow can occur in arithmetic operations and ASL. Similar to carry, it is computed in each situation where overflow can occur. A MUX selects relevant cases. We give out a four-bit flag output and connected it to a 4-bit register in 'part3-alu'.

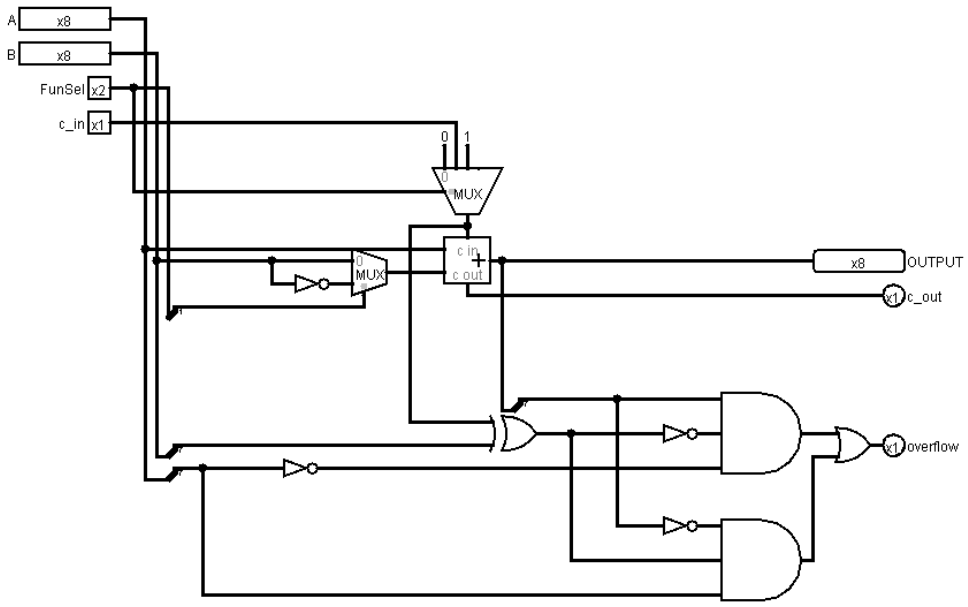

Figure 5: Logisim Circuit For Part-3 Arithmetic Operation

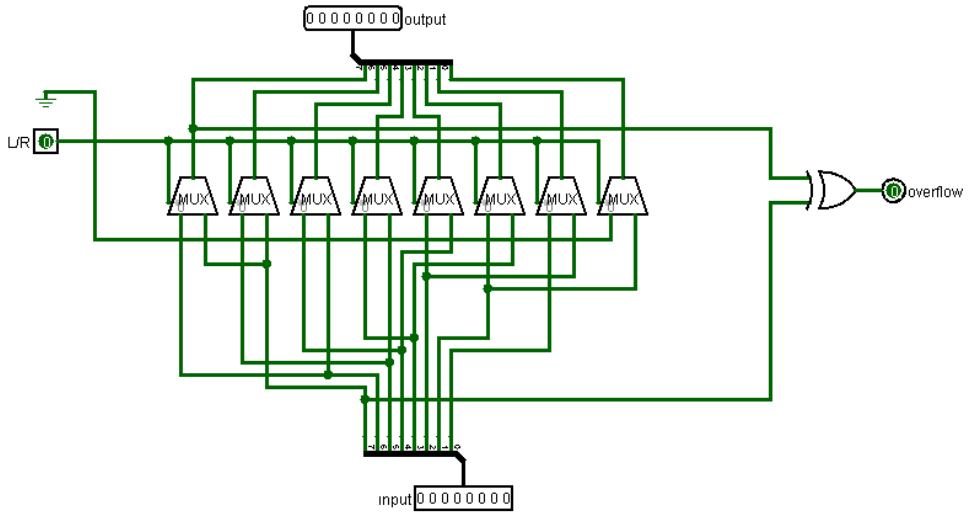

Figure 6: Logisim Circuit For Part-3 Arithmetic Shift

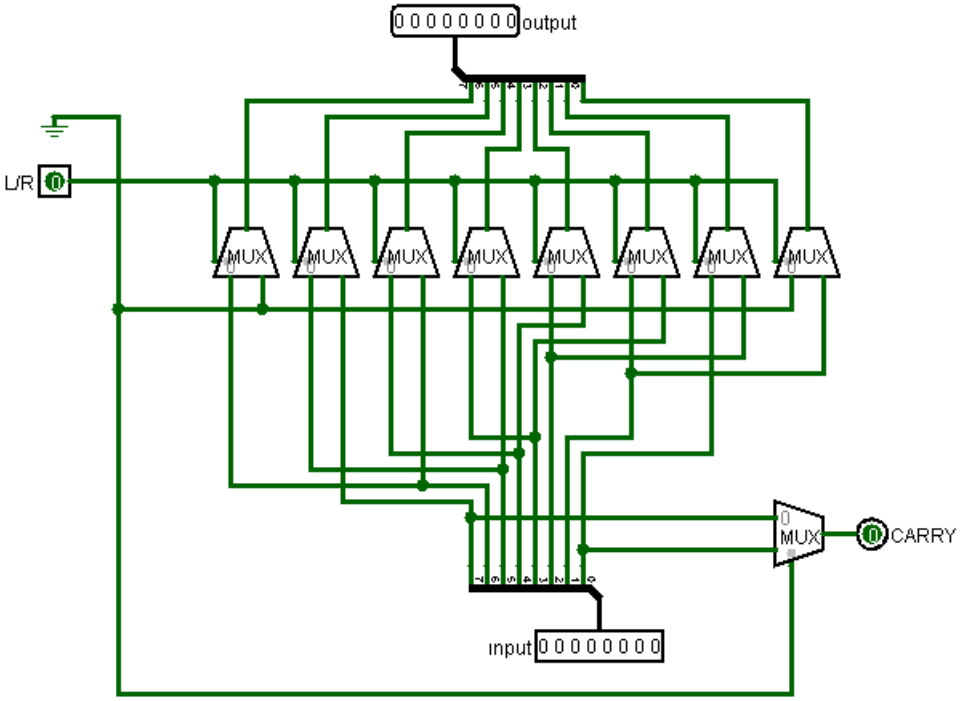

Figure 7: Logisim Circuit For Part-3 Logical Shift

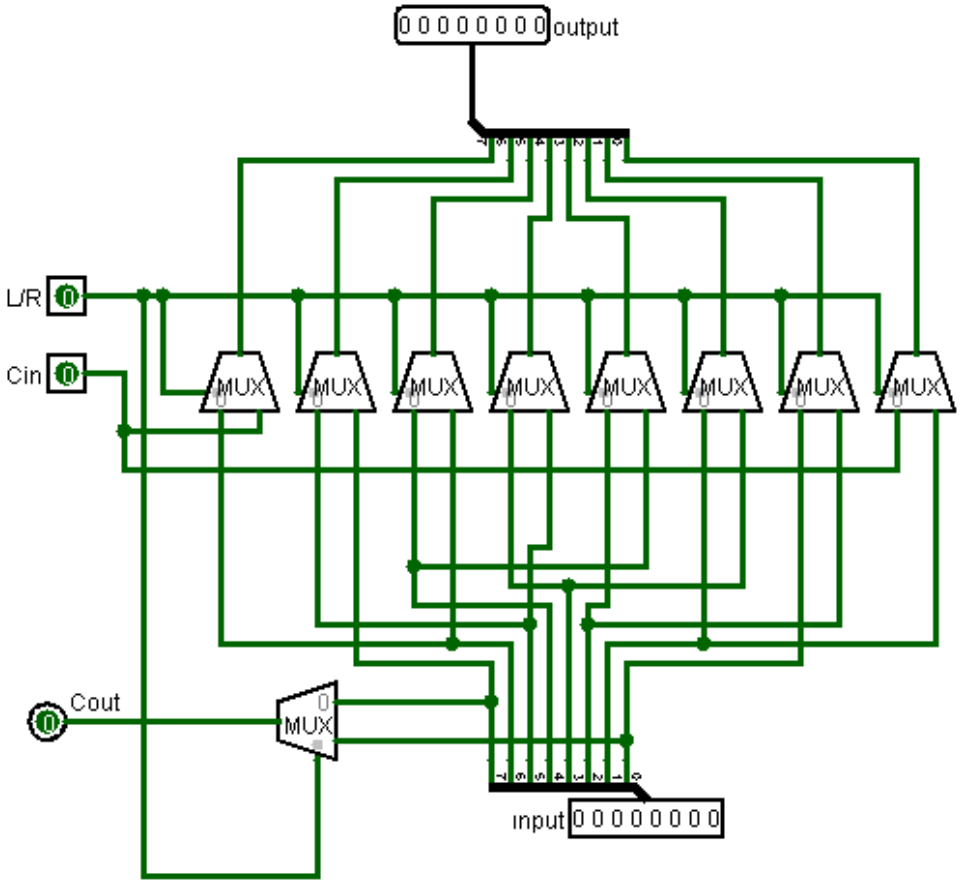

Figure 8: Logisim Circuit For Part-3 Circular Shift

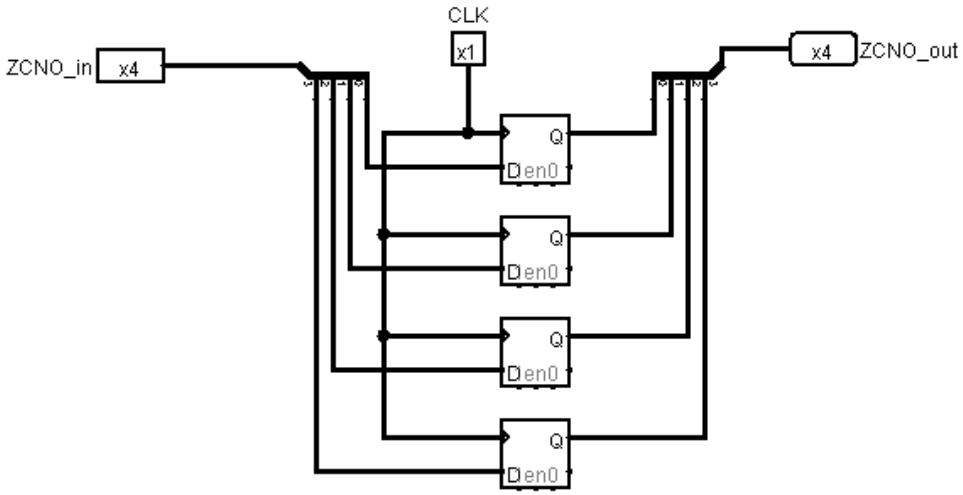

Figure 9: Logisim Circuit For Part-3 Flag

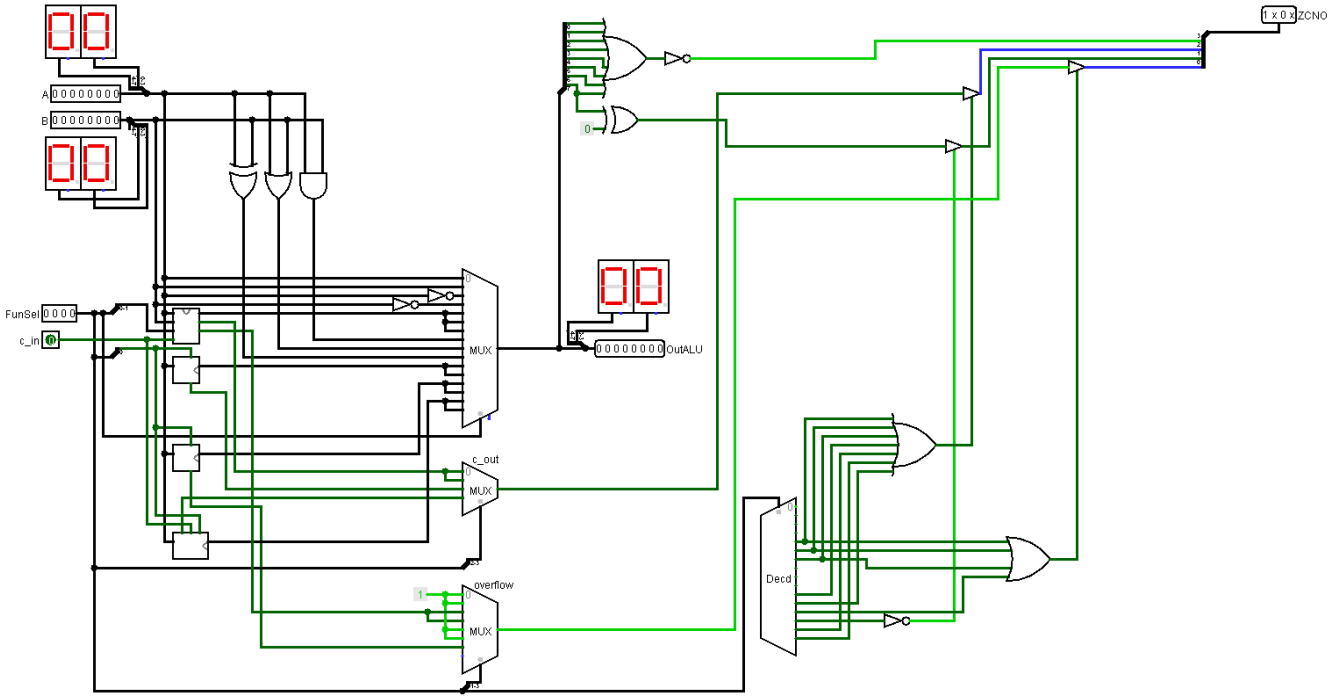

Figure 10: Logisim Circuit For Part-3 ALU

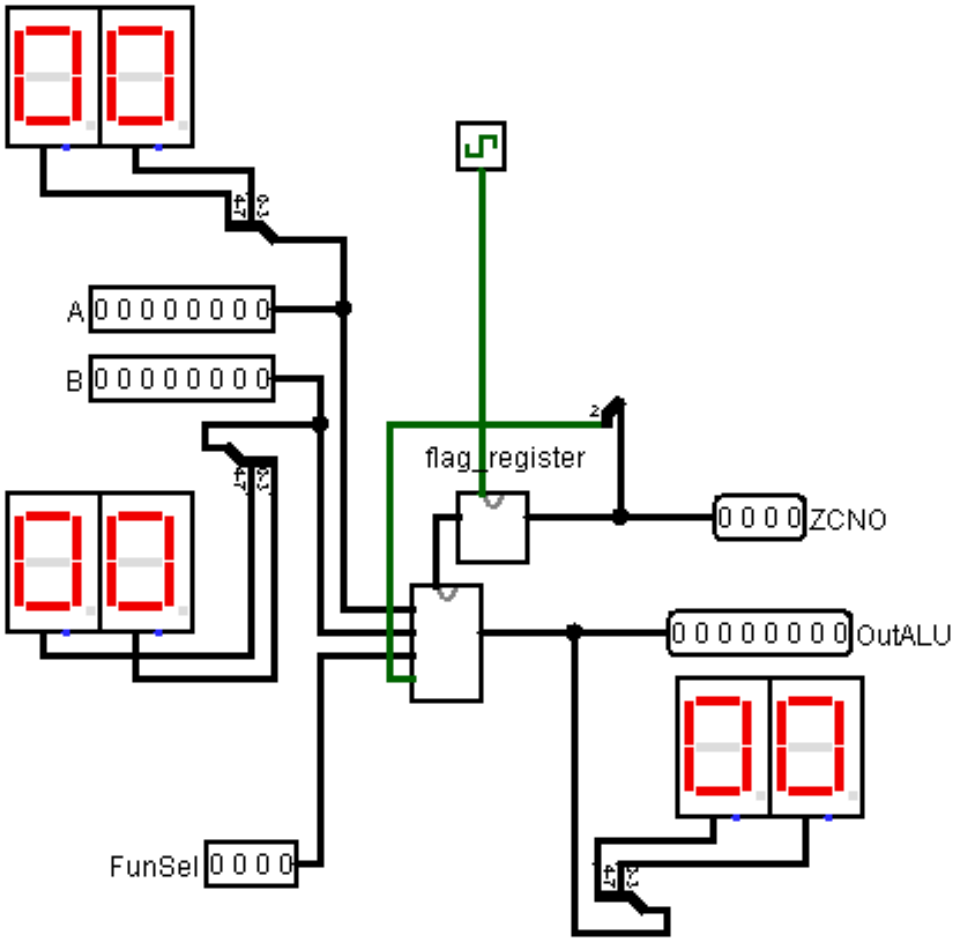

Figure 11: Logisim Circuit For Part-3 Main

2.4 PART 4

As shown in Figure 12, there are following components in our design:

- Register File(Part2a): We used register file which we designed earlier for Part2a. 8-bit input I comes from Multiplexer B. Inputs 4-bit RegSel and 4-bit TmpSel select the general-purpose and temporary register respectively going to be activated. Inputs 3-bit OutASel and 3-bit OutBSel select for feed the output lines OutA and OutB respectively. Input 2-bit FunSel selects which operation going to implement according to FunSel Control Input Table in Part2a. Outputs 8-bit OutA and 8-bit OutB connected to ALU.

- ALU(Part3): We used Arithmetic Logic Unit (ALU) which we designed earlier for Part3. ALU takes the outputs of the register file as inputs. Input 4-bit FunSel selects which operation going to implement according charachteristic table of ALU in Part3. 4-bit output ZCNO connected to Flag Register to store latest value of ZCNO. 1-bit input c_{in} comes from Flag Register and represents C. OutALU connected to

memory(RAM) and both multiplexers.

- Flag Register: We designed and used 4-bit Flag Register. It takes ZCNO value from ALU and stores in itselves. We used this unit for taking Carry(C) as a input for ALU at each and every clock signal.

- Address Register File (ARF)(Part2b): We used Address Register File (ARF) which we designed earlier for Part2b. 8-bit input I comes from output of the Multiplexer B. 3-bit input RegSel selects which register going to enabled in ARF. 2-bit input FunSel selects which operation going to be implement according to Funsel Control Input Table in Part2a. Inputs 3-bit OutCSel and 3-bit OutDSel select for feed the output lines OutC and OutD respectively. 8-bit output OutC is connected to both multiplexers. 8-bit OutD is connected to the RAM.

- Memory(RAM): We used RAM unit. 8-bit Address input OutD comes from ARF. 8-bit input value to be stored at address comes ALU. Each and every clock signal, data loaded from the address (memory output) goes both multiplexers as inputs.

- 16-bit IR Register(Part1b): We used 16-bit IR Register which we designed earlier for Part1b. 8-bit input of IR Register comes from in Memory(RAM) unit which is data loaded from address. Inputs 1-bit L/H, 1-bit ENABLE, 2-bit FunSel work according to characteristic table of Part1b. LSB output of IR register(0-7) connected to both multiplexers.

- Multiplexer A: We used 4:1 multiplexer. Input MuxASel connected to Multiplexer A and selects the MuxAOut. 8-bit MuxAOut connected to Register File as input I.

- Multiplexer B: We used 4:1 multiplexer. Input MuxBSel connected to Multiplexer B and selects the MuxBOut. 8-bit MuxBOut connected to Address Register File (ARF) as input I.

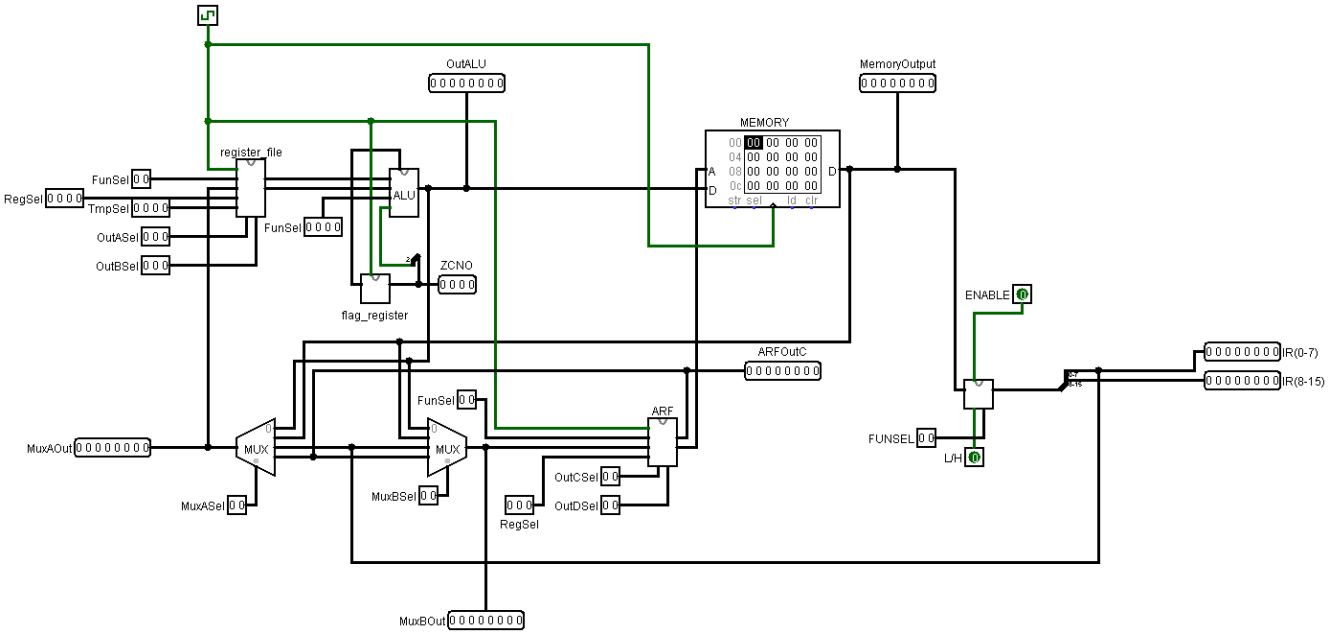

Figure 12: Logisim Circuit For Part-4

3 RESULTS

We want to pick outB in the register as R2 because we want to load R1 with NOT (R2), so OutBSel should be 001. Since it should not be B, our ALU procedure should be FunSel 0011. Memory and MuxA receive the meaning from the ALU. MuxASel must be 0 and OutALU must be subtracted. Then OutALU becomes the Register File's input. The register's FunSel becomes 10, indicating that the load has completed, and the RegSel becomes 0111, indicating that our value has loaded into R1.

We want to load R3 with LSR(R1), and we want outA in the register to be R1, so outASel should be 000, and the ALU's A input will be 0xFF. The FunSel of the ALU will be 1011, allowing us to perform LSR on the A input. The ALU output will be sent to MuxASel, which is set to 00, and it will pick the ALU output and pass it to Register File as input. We'll load LSR(R1) into R3, so we'll set the RegSel of register file 1101 to 10 and the Funsel of register file to 10.

We'll set OutASel of the register file to 010 (R3 is selected) to load SP with R3's value, so A input of the ALU will be 0x7F. We'll set ALU's FunSel to 0000 to keep things the same. The output of the ALU will be sent to MuxB, and we'll set MuxBSel to 00, causing MuxB to output OutALU. The output of MuxB is the address register file's input. The ARF Regsel is 110 to enable SP, and the ARF Funsel is 10 to enable LOAD operation.

SP will contain the R3 value of 0x7F after the clock is activated.

4 DISCUSSION

Throughout the project, we designed step-by-step circuits mainly to build the ALU System. We combined the circuits we created with connections as we explained in PART 4. While designing this project, we used nested modules in some parties. For example, in Part 3, we designed the shift operations as separate circuits and used them as modules so that the main circuit does not appear mixed, just as we used all the circuits we designed in the last part as modules and connected each other. Our modules, which receive their inputs from the output of another circuit, receive the CLK signal and start working.

5 CONCLUSION

In this project, we designed 8-bit and 16-bit register, register file, Arithmetic Logic Unit, and the organization given in the document. When we were doing this project, we learned how the ALU works and we took ourselves to the next level in using the Logisim program. Although we were confused about how to create parts at first and use them in other parts, over time we got used to this situation.