ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT REPORT

PROJECT NO : 1

DUE DATE : 19.05.2021

GROUP NO : G55

GROUP MEMBERS:

150190710 : Serra Bozkurt

150190063 : Gökalp Akartepe

150190024 : Ahmet Furkan Kavraz

1 INTRODUCTION

In this homework, we are asked to implement different types of registers, register file, address register file, Arithmetic Logic Unit and at last a system consists of these components with a same clock signal.

For these implementations, we have reviewed our knowledge of registers, register files, ALU and Logisim program. We also reviewed the course slides which were really helpful. After all, we implemented the modules one by one according to the expectations in the homework documentation.

All circuits are made according to the tables in the given documentation.

Then, we tested these circuits wire different test cases. Finally, we checked the results to see if the outputs match our expectations.

2 PROJECT PARTS

2.1 PART 1

2.1.1 PART 1A

As shown in Figure 1, there are following components in our 8 Bit Register, Part-1a design:

- 8-Bit D FlipFlop: We use 8-Bit D FlipFlop for storing 8 Bit Data.
- Adder: We use adder to obtain Q+1 value.
- Subtractor: We use subtractor to obtain Q-1 value.
- Constant: We use constant 1 for giving 8-bit input to adder and subtractor.
- Multiplexer: We use multiplexer for choosing the correct value among Q, Q-1, Q+1, Input, and 0 values.

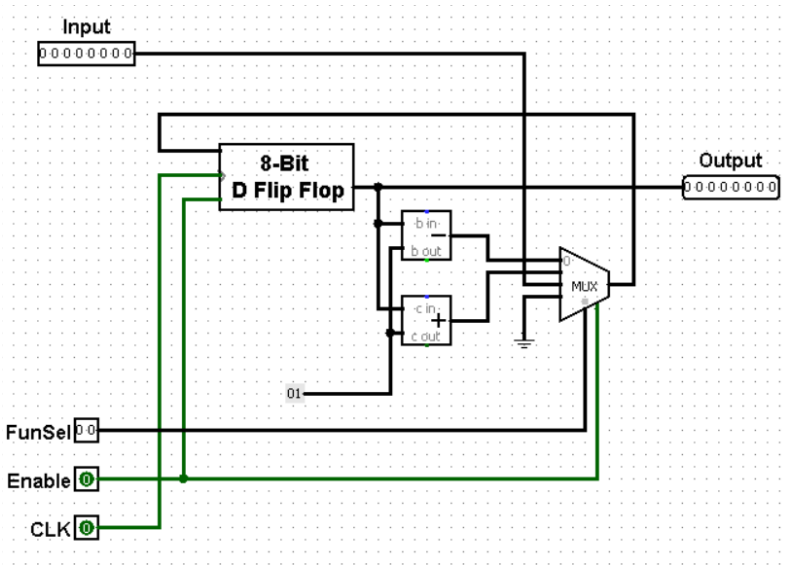

Figure 1: Logisim Circuit For 8 Bit Register, Part-1a

As shown in Figure 2, there are following components in our 8 Bit D FlipFlop, Part-1a design:

- D FlipFlop: We use D FlipFlop for storing 1 Bit Data.
- Splitters: We use splitters, one is for separating the input value and giving them D FlipFlop as input and the other one is for combining the output values of D FlipFlops and creating Output.

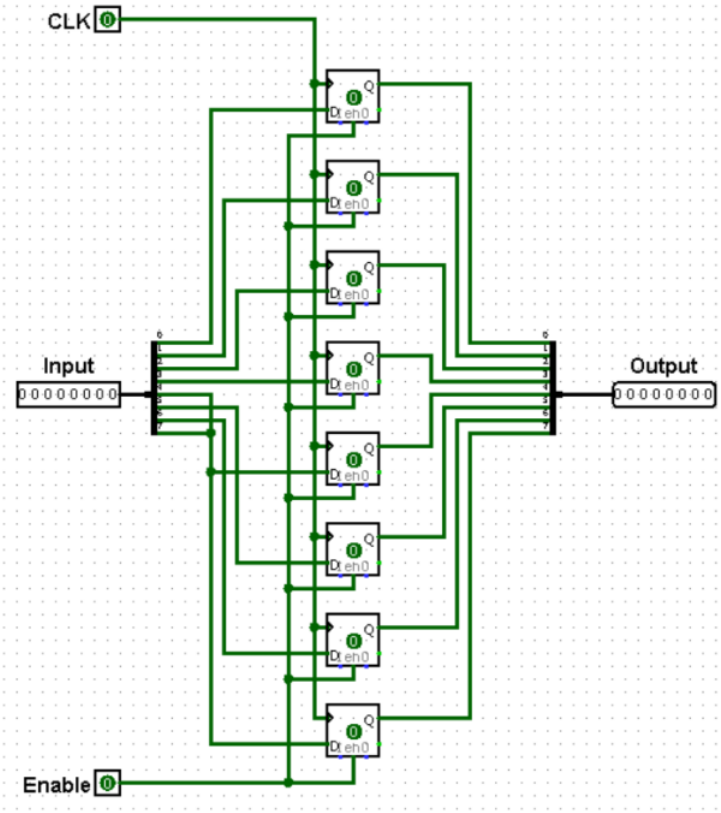

Figure 2: Logisim Circuit For 8 Bit D FlipFlop, Part-1a

2.1.2 PART 1B

As shown in Figure 6, there are following components in our design:

- Input: We use input input components for taking a 8-bit data, taking control signals and clock signal.
- Splitter: We use splitters in order to split incoming data into the parts.
- DeMultiplexer: We use this demultiplexer for giving appropriate inputs into the 16 bit register that we implemented.
- 16-bit Register: This component is used to calculate the output according to the inputs from demultiplexer, L/H and FunSel. It decides which operation will be made, operates, and yields the 16-bit output. And yields the final result, according to the control and data inputs.
- Hex-Digit Display: We use it for monitoring our final result.
- 8-bit D Flip Flop: This component is used for combining outputs from 8 different 1-bit D type flip flops, hence it is a 8-bit D flip flop.

- 16-bit D Flip Flop: This component is used for obtaining outputs of 2 different 8-bit D type flip flops, hence it is a 16-bit D flip flop.

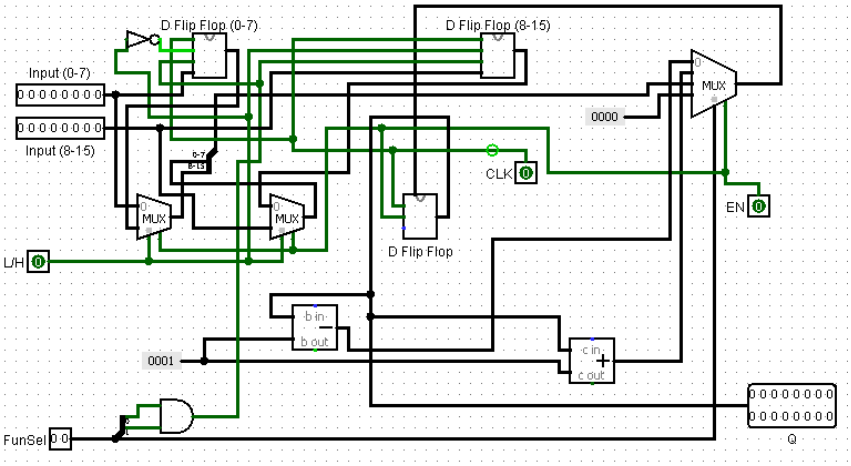

Figure 3: Logisim Circuit For 16-bit Register for Part1-B

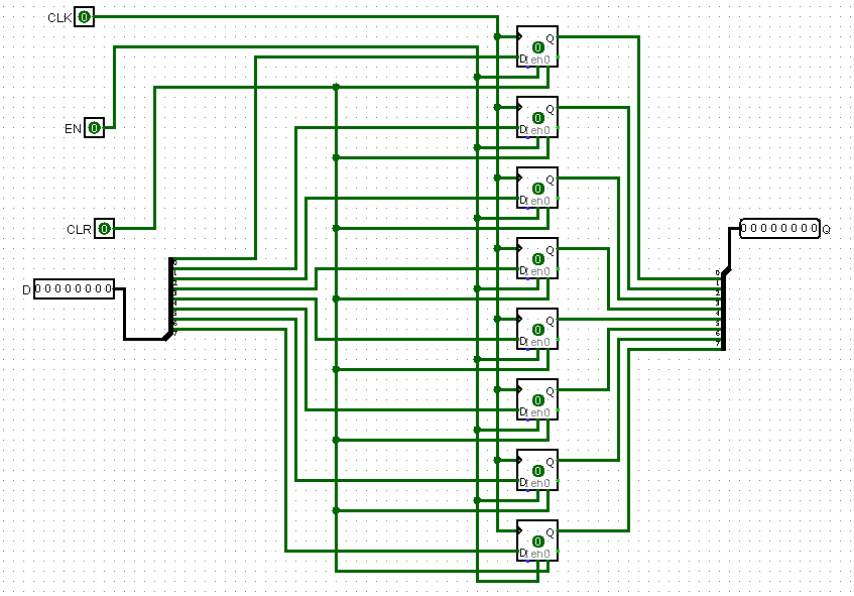

Figure 4: Logisim Circuit Of 8-bit D Flip Flop for Part-1b

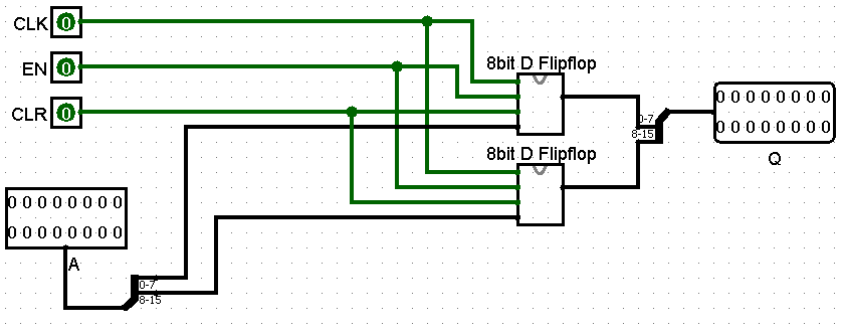

Figure 5: Logisim Circuit Of 16-bit D Flip Flop for Part-1b

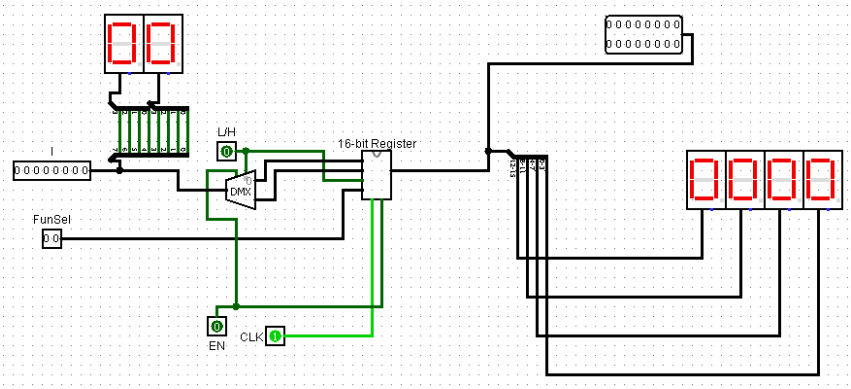

Figure 6: Logisim Circuit For Part-1b

2.2 PART 2

2.2.1 Part-2.A

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

- **8-bit Registers:** We used 8-bit registers we designed in Part-1.A, to perform different operations on the data we have, like storing the information, incrementing/decrementing the value, or clearing it. We needed something to decide whether to enable them or not, so we used RegSel and TmpSel inputs in the following manner:

- **RegSel and TmpSel inputs(4-bit input):** Both of them are 4-bit inputs that we use to choose which 8-bit register/registers will be enabled and which are not. So both will be connected to the corresponding registers they control.

  - RegSel is used in the 8-bit registers from Part-1.A to decide which of them will be enabled and will not be enabled. The relationship between RegSel and General Purpose 8-bit registers are given to us in the pdfs, which can be seen in below Figure 7. The summarised rule is this:

    * If 0^{th} bit of RegSel = A, Enable of R4 = A'.

* If 1^{st} bit of RegSel = B, Enable of R3 = B'.
* If 2^{nd} bit of RegSel = C, Enable of R2 = C'.
* If 3^{rd} bit of RegSel = D, Enable of R1 = D'.

– TmpSel is used in the Temporary 8-bit registers to decide which of them will be enabled and will not be enabled. The relationship between TmpSel and Temporary 8-bit registers are given to us in the pdfs, which can be seen in below Figure 8

* If 0^{th} bit of TmpSel = A, Enable of T4 = A'.
* If 1^{st} bit of TmpSel = B, Enable of T3 = B'.
* If 2^{nd} bit of TmpSel = C, Enable of T2 = C'.
* If 3^{rd} bit of TmpSel = D, Enable of T1 = D'.

– Both the inputs are basically to enable between the given 8-bit registers, namely R1, R2, R3, R4, T1, T2, T3, T4.

RegSel	Enabled General Purpose Registers
0000	ALL general purpose registers are enabled (Function selected by FunSel will be applied to R1, R2, R3 and R4)
0001	R1, R2 and R3 are enabled (Function selected by FunSel will be applied to R1, R2 and R3)
0010	R1, R2 and R4 are enabled, Function selected by FunSel will be applied to R1, R2 and R4
0011	R1 and R2 are enabled (Function selected by FunSel will be applied to R1 and R2)
0100	R1, R3 and R4 are enabled (Function selected by FunSel will be applied to R1, R3 and R4)
0101	R1 and R3 are enabled (Function selected by FunSel will be applied to R1 and R3)
0110	R1 and R4 are enabled (Function selected by FunSel will be applied to R1 and R4)
0111	Only R1 is enabled (Function selected by FunSel will be applied to R1)
1000	R2, R3 and R4 are enabled (Function selected by FunSel will be applied to R2, R3 and R4)
1001	R2 and R3 are enabled (Function selected by FunSel will be applied to R2 and R3)
1010	R2 and R4 are enabled (Function selected by FunSel will be applied to R2 and R4)
1011	Only R2 is enabled (Function selected by FunSel will be applied to R2)
1100	R3 and R4 are enabled (Function selected by FunSel will be applied to R3 and R4)
1101	Only R3 is enabled (Function selected by FunSel will be applied to R3)
1110	Only R4 is enabled (Function selected by FunSel will be applied to R4)
1111	N0 general purpose register is enabled (All R1, R2, R3 and R4 registers retain their values)

Figure 7: Relationship between RegSel and General Purpose 8-bit registers

- Not Gates: We used not gates to obtain the RegSel (4-bit input) and TmpSel (4-bit input) input values' complement, wrt 1's complement notation. We needed the complement for the following reason: We need all the bits of both RegSel and TmpSel inputs' complement, as an enable for the 8-bit registers, explained in the above item.

TmpSel	Enabled Temporary Registers
0000	ALL temporary registers are enabled (Function selected by FunSel will be applied to T1, T2, T3 and T4)
0001	T1, T2 and T3 are enabled (Function selected by FunSel will be applied to T1, T2 and T3)
0010	T1, T2 and T4 are enabled, Function selected by FunSel will be applied to T1, T2 and T4
0011	T1 and T2 are enabled (Function selected by FunSel will be applied to T1 and T2)
0100	T1, T3 and T4 are enabled (Function selected by FunSel will be applied to T1, T3 and T4)
0101	T1 and T3 are enabled (Function selected by FunSel will be applied to T1 and T3)
0110	T1 and T4 are enabled (Function selected by FunSel will be applied to T1 and T4)
0111	Only T1 is enabled (Function selected by FunSel will be applied to T1)
1000	T2, T3 and T4 are enabled (Function selected by FunSel will be applied to T2, T3 and T4)
1001	T2 and T3 are enabled (Function selected by FunSel will be applied to T2 and T3)
1010	T2 and T4 are enabled (Function selected by FunSel will be applied to T2 and T4)
1011	Only T2 is enabled (Function selected by FunSel will be applied to T2)
1100	T3 and T4 are enabled (Function selected by FunSel will be applied to T3 and T4)
1101	Only T3 is enabled (Function selected by FunSel will be applied to T3)
1110	Only T4 is enabled (Function selected by FunSel will be applied to T4)
1111	N0 temporary register is enabled (All T1, T2, T3 and T4 registers retain their values)

Figure 8: Relationship between TmpSel and Temporary 8-bit registers

- Multiplexers (2 different MUX, selections made by OutASel and OutBSel): We use these to select the appropriate output between different 8-bit registers we used, Temporary ones and General-Purpose ones. You can see the working principle of the multiplexer according to the given selectors in the Figure 9. We needed these 8:1 multiplexers to choose the output among 8 inputs.
  - OutASel is used in the upper multiplexer to decide the OutA output value.
  - OutBSel is used in the lower multiplexer to decide the OutB output value.
  - Both the multiplexers choose between all the given 8-bit registers, namely R1, R2, R3, R4, T1, T2, T3, T4.

- **FunSel (2-bit input):** We used FunSel input to decide what to do with the information we have in registers. The performable operations are given to us in the pdf as:
  - Load
  - Clear
  - Increment
  - Decrement

  and also have been put in the following Figure 10.

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	T1
101	T2
110	T3
111	T4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	T1
101	T2
110	T3
111	T4

Figure 9: Selection Table, using OutASel and OutBSel

FunSel	R_x^+	
00	R_x-1	(Decrement)
01	R_x+1	(Increment)
10	I	(Load)
11	0	(Clear)

Figure 10: Selection Table, using FunSel

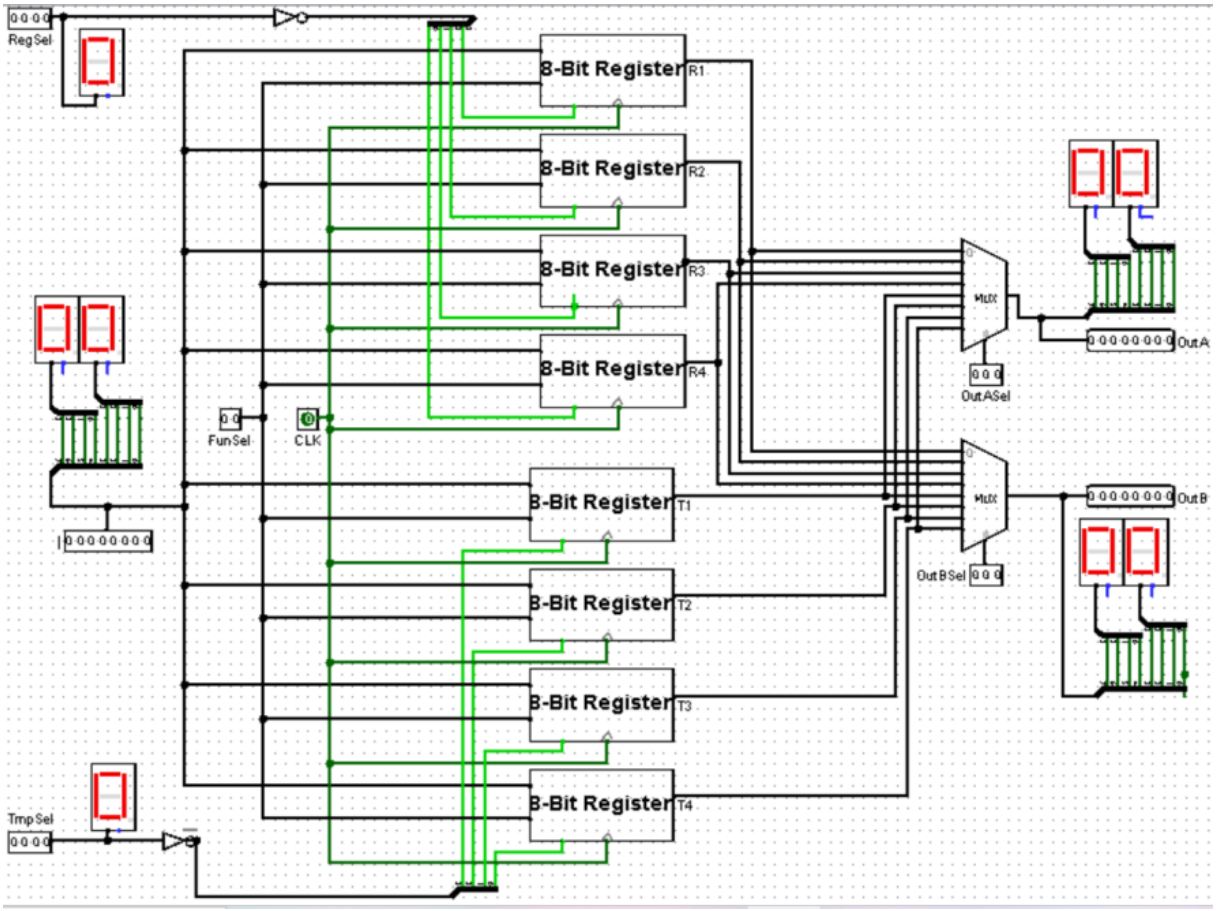

Figure 11: Logisim Circuit Of Part-2.A

2.2.2 Part-2.B

As shown in Figure 13, there are following components in our design:

- **8-bit Registers:** We used 3 8-bit registers (SP, PC, AR which are stand for Stack Pointer, Program Counter and Address Register) we designed in Part-1.A, to perform different operations on the data we have, like storing the information, incrementing/decrementing the value, or clearing it. We needed something to decide whether to enable them or not, so we used RegSel input in the following manner:

- **RegSel input(3-bit input):** input that we use to choose which 8-bit register/registers will be enabled and which are not. So it will be connected to the corresponding registers its' bits control. The relationship between RegSel and 8-bit registers are given to us in the pdfs as **"same with Part-2.A"**. So the summarised rule is:

  - If 0^{th} bit of RegSel = A, Enable of SP = A'.
  - If 1^{st} bit of RegSel = B, Enable of AR = B'.
  - If 2^{nd} bit of RegSel = C, Enable of PC = C'.

- Not Gate: We used not gate to obtain the RegSel (3-bit input) input's complement, according to 1's complement notation. We needed the complement for the following reason: We need all the bits of RegSel's complement, as an enable for the 8-bit registers, explained in the above item.

- Multiplexers (2 different MUX, selections made by OutCSel and OutDSel): We use these to select the appropriate output between different 8-bit registers we used, Temporary ones and Part-1.A ones. You can see the working principles of the multiplexers according to the given selectors in the Figure 12. We needed these 4:1 multiplexers to choose the output among 4 inputs.

  - OutCSel is used in the upper multiplexer to decide the OutC output value.
  - OutDSel is used in the lower multiplexer to decide the OutD output value.
  - Both the multiplexers choose between all the given 8-bit registers, namely SP, PC, AR.

- **FunSel (2-bit input):** We used FunSel input to decide what to do with the information we have in registers. The performable operations are given to us in the pdf as: Load, Clear, Increment, Decrement. The relationship between FunSel and 8-bit registers are given to us in the pdfs as **"same with Part-2.A"**. So have a look at Figure 10.

OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

OutDSel	OutD
00	PC
01	PC
10	AR
11	SP

Figure 12: Selection Table for multiplexers, using OutCSel and OutDSel

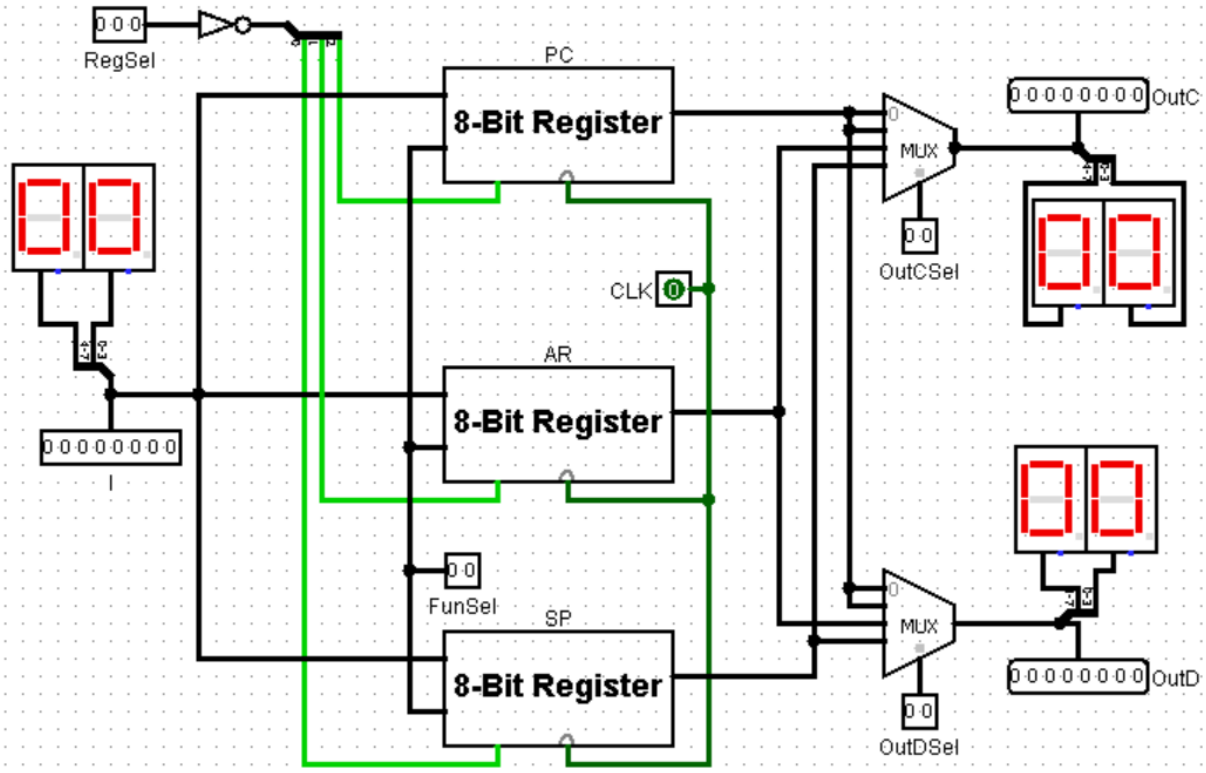

Figure 13: Logisim Circuit Of Part-2.B

2.3 PART 3

As shown in Figure 14, there are following components in our ALU design:

- Splitters: We use splitters for giving correct value of FunSel values to Units. First two or three bit of FunSel is given to units.
- Buffer-Not Unit: It is used for performing Buffer and Not Gate operations.
- Aritmetic Unit: It is used for performing Addition, Addition with Carry and Subtraction operations.
- Logic Unit: It is used for performing And, Or and Xor operations.
- Shift Unit: It is used for performing Logical Right/Left Shift, Arithmetic Right/Left Shift and Circular Right/Left Shift.
- All Units that we used gives two Output. One is OutALU value and the other is OutFlag value.
- Multiplexers: We use multiplexers to choose the correct output of the units for values outALU and outFlag.

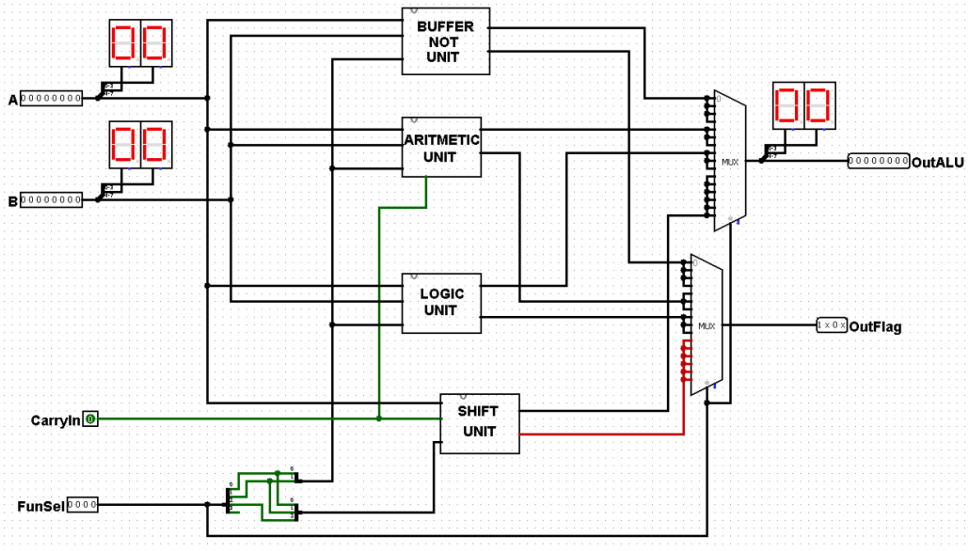

Figure 14: Logisim Circuit For Part-3

As shown in Figure 15, there are following components in our Buffer-Not Unit design:

- Not Gates: We use Not gates to obtain complements of inputs.
- Multiplexer: We use multiplexer to select correct output for OutALU.
- Nor Gate: We use Nor gate to obtain Zero Flag.
- Splitter: We use splitter to combine flags and obtain OutFlag output.

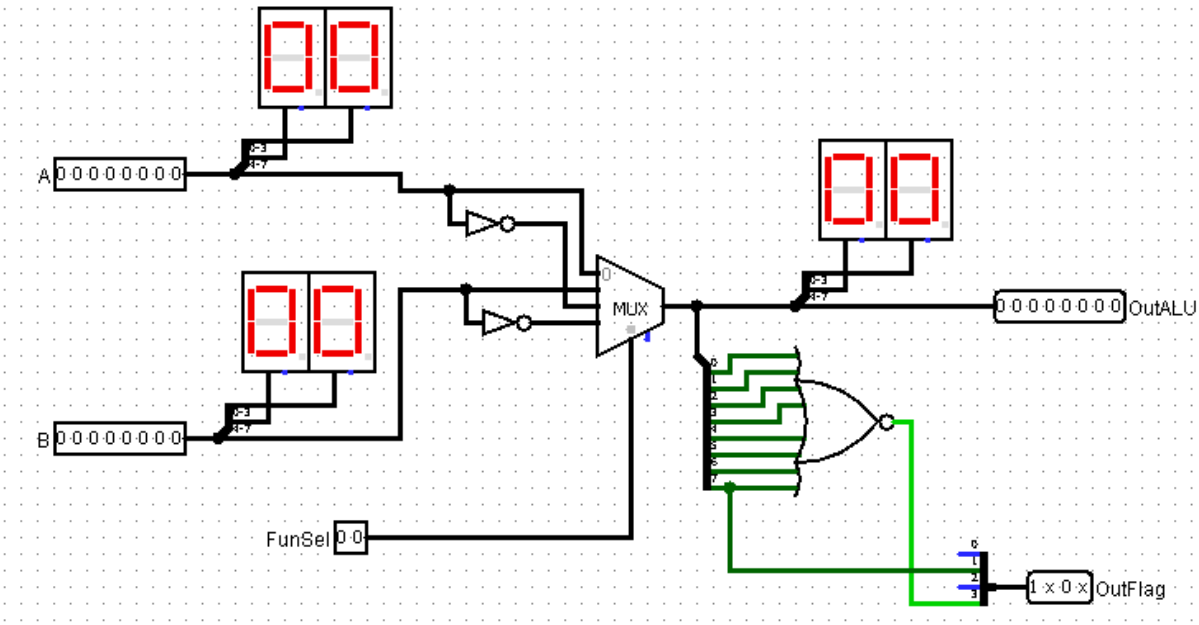

Figure 15: Logisim Circuit of Buffer-Not Unit for Part-3

As shown in Figure 16, there are following components in our Arithmetic Unit design:

- Adder: We use adder to obtain A+B, with or without carry input, and carry flag according to FunSel.
- Subtractor: We use subtractor to obtain A-B and carry flag.
- Multiplexers: We use multiplexers for choosing correct OutALU output, Carry and Overflow flags according to FunSel.
- And and Or Gates: We use And and Or gates to obtain overflow flags and connect the outputs to multiplexer to choose correct one.
- Nor Gate: We use Nor gate to obtain Zero flag.
- Not Gates: We use Not gates to obtain complements of borrow output of subtractor(it is equal to carry), complement of most significant bits of inputs and OutALU.

- Splitters: We use splitters to obtain last bits of inputs, first bit of FunSel, bits of OutALU and combine flags for correct OutFlag output.
- And Gate: The And gate, that is in left most side of the circuit, is used for getting adder output with or without carry input.

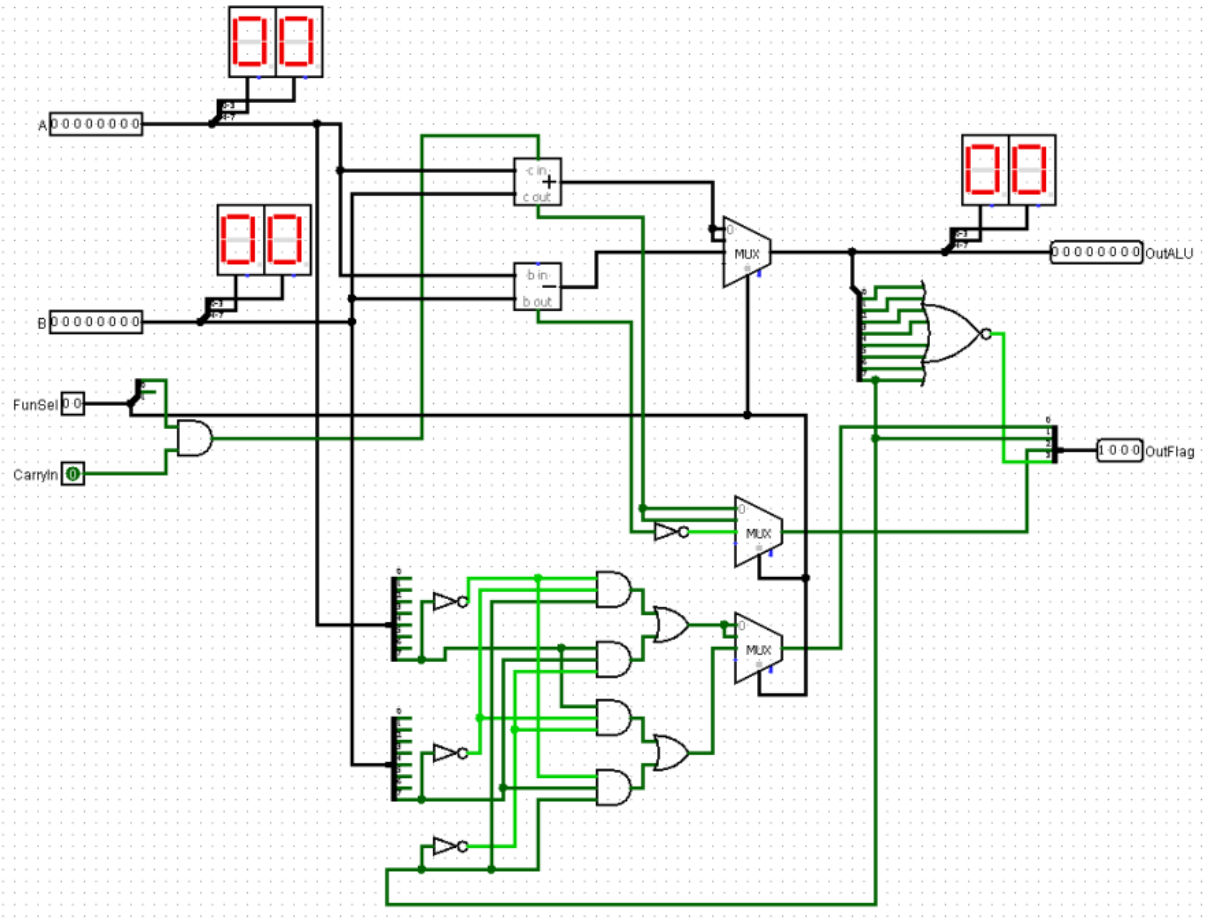

Figure 16: Logisim Circuit of Arithmetic Unit for Part-3

As shown in Figure 17, there are following components in our Logic Unit design:

- And Gates: We use And gates to obtain A (and) B.
- Or Gates: We use Or gates to obtain A (or) B.
- Xor Gates: We use Xor gates to obtain A (xor) B.
- Nor Gate: We use Nor gate to obtain Zero Output.
- Multiplexer: We use multiplexer to select correct output for OutALU according to FunSel.
- Splitter: We use splitter to combine flags and obtain correct OutFlag output.

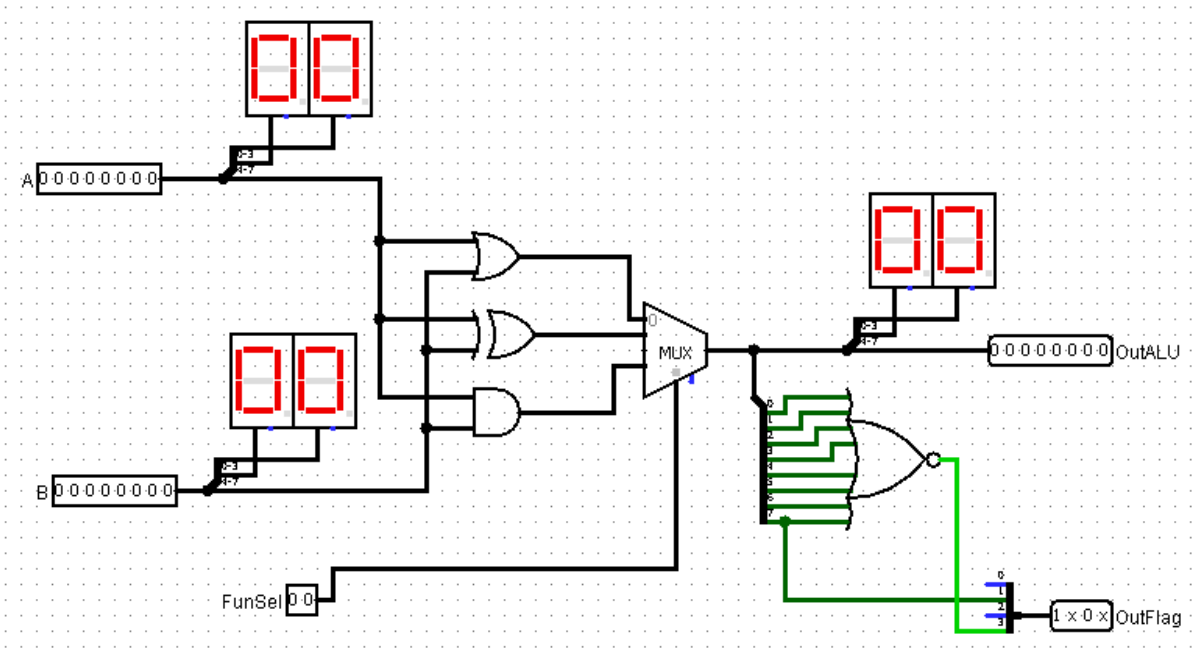

Figure 17: Logisim Circuit of Logic Unit for Part-3

As shown in Figure 18, there are following components in our Shift Unit design:

- Logical Left Shifter: We use logical left shift to obtain logical shift left and Arithmetic Shift Left of input.
- Logical Right Shifter: We use logical right shifter to obtain Logical Shift Right of input.
- Arithmetic Right Shifter: We use arithmetic right shifter to obtain Arithmetic Shift Right of input.
- Splitters: We use splitter, which is in middle part of circuit, to obtain Circular Left and Right Shifts of input.
- Multiplexer: We use multiplexers to select correct output for OutALU.
- Multiplexers Splitter Xor Gates: We use multiplexers, splitter and xor gates to obtain carry and overflow flags.
- Nor Gate: We use Nor gate to obtain Zero Output.
- Splitter: We use splitter, which is in right most side of circuit, to obtain correct OutFlag output.

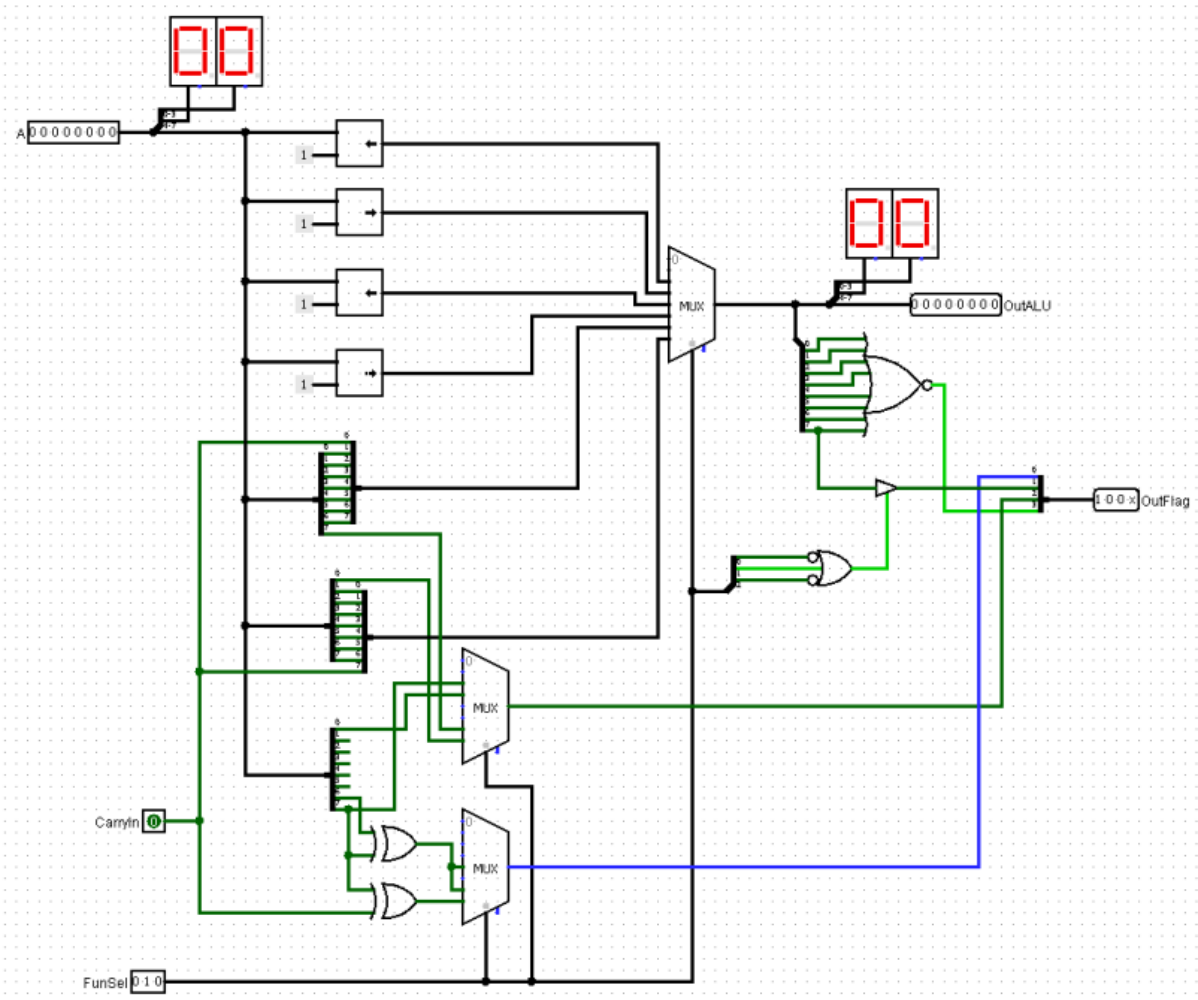

Figure 18: Logisim Circuit of Shift Unit for Part-3

2.4 PART 4

As shown in Figure 19, there are following components in our design:

- Multiplexer: We use the multiplexers to choose among the data values coming from registers, ARF and ALU.
- ALU: We use ALU component for performing operations.
- ARF: We use ARF component to decide which part of the circuit, the data will be given, according to OutDSel, OutCSel etc., such as PC, AR and SP.
- Register File: We use register file component for deciding which register we will hold the data, and make the necessary operations on the input according to the corresponding input values RegSel, FunSel and TmpSel, given in the table from the experiment documentation.
- IR: We use IR 16-bit register for loading the data to its LSB or MSB, or increment/decrement the value or clearing it according to L/H, FUNSEL and EN inputs, as we are asked in the experiment documentation.
- Memory: We use the memory for storing the necessary values, according to its inputs, for our circuit.

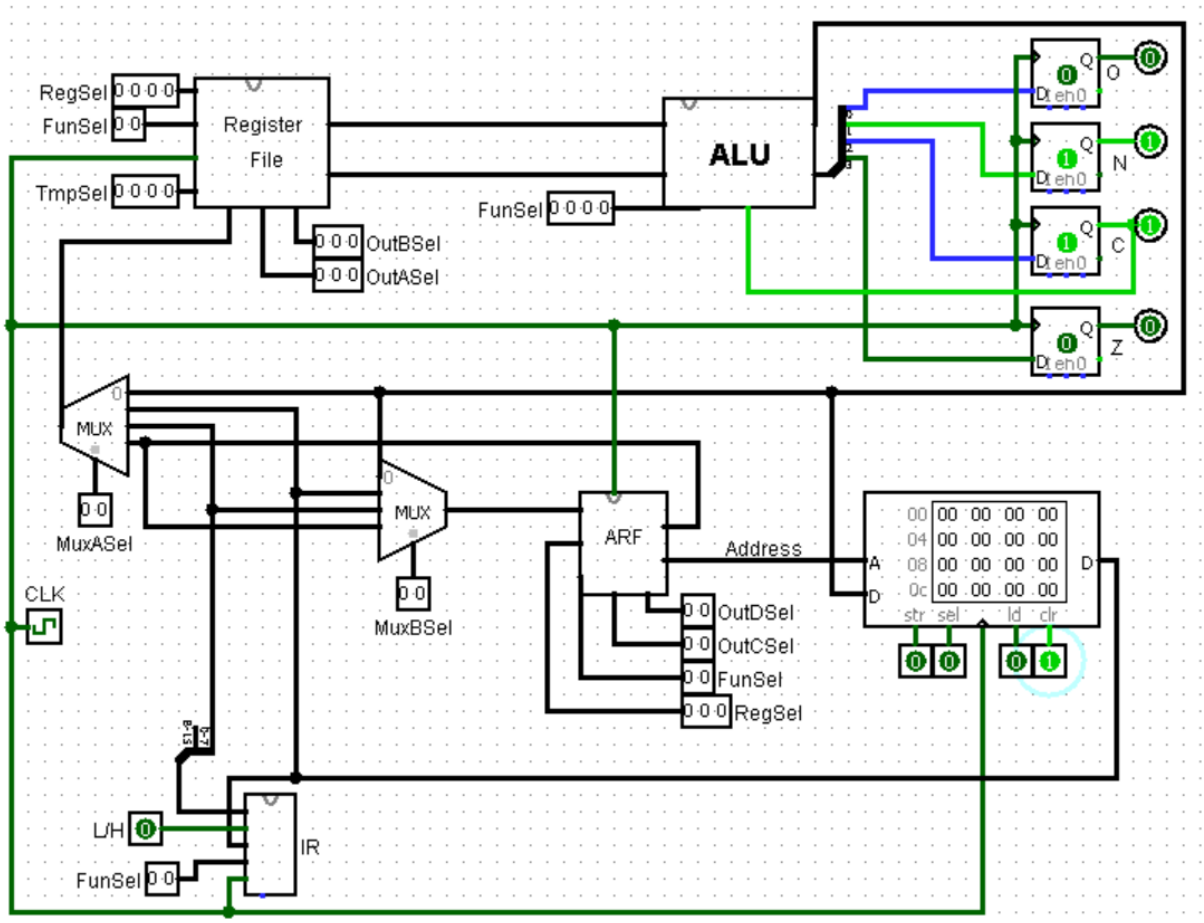

Figure 19: Logisim Circuit For Part-4

3 RESULTS

We tried our implementations with the given test cases from ninova.

- For the 1st test case, our aim is to Load Register File R1 with NOT(R2) = R2' value. You can see the clear implementation in Figure 20 Our input values will be in the following manner:
  - RegSel of Register File: 0111
  - TmpSel of Register File: 0000
  - FunSel of Register File: 10
  - OutBSel of Register File: 001
  - OutASel of Register File: 000
  - FunSel of ALU: 0011
  - RegSel of Address Register File: 000
  - FunSel of Address Register File: 00
  - OutCSel of Address Register File: 00
  - OutDSel of Address Register File: 00
  - MuxASel: 00
  - MuxBSel: 00
- And if we enter 1 full clock cycle, we can see our R1 = 0xFF, N = 1, C = 0 in the following Figure 20.
- And if we enter the ALU file during the simulation, we can see our ALU output = 0x7F, in the following Figure 21.
- For the 2nd test case, our aim is to Load Register File R3 with LSR(R1) value. You can see the clear implementation in Figure 22 Our input values will be in the following manner:
  - RegSel of Register File: 1101
  - TmpSel of Register File: 0000
  - FunSel of Register File: 10
  - OutASel of Register File: 000
  - OutBSel of Register File: 001
  - FunSel of ALU: 1011

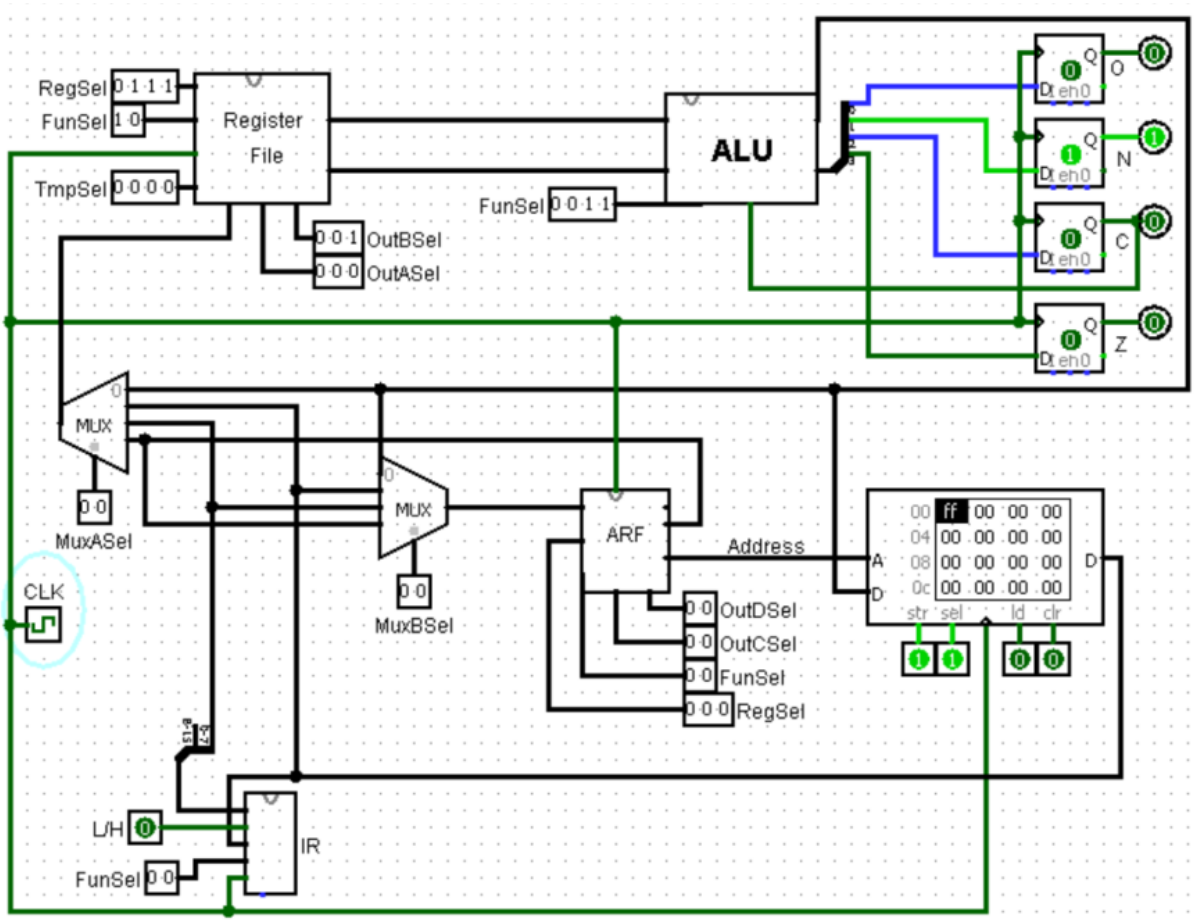

Figure 20: Logisim Circuit while applying first test case datas to design

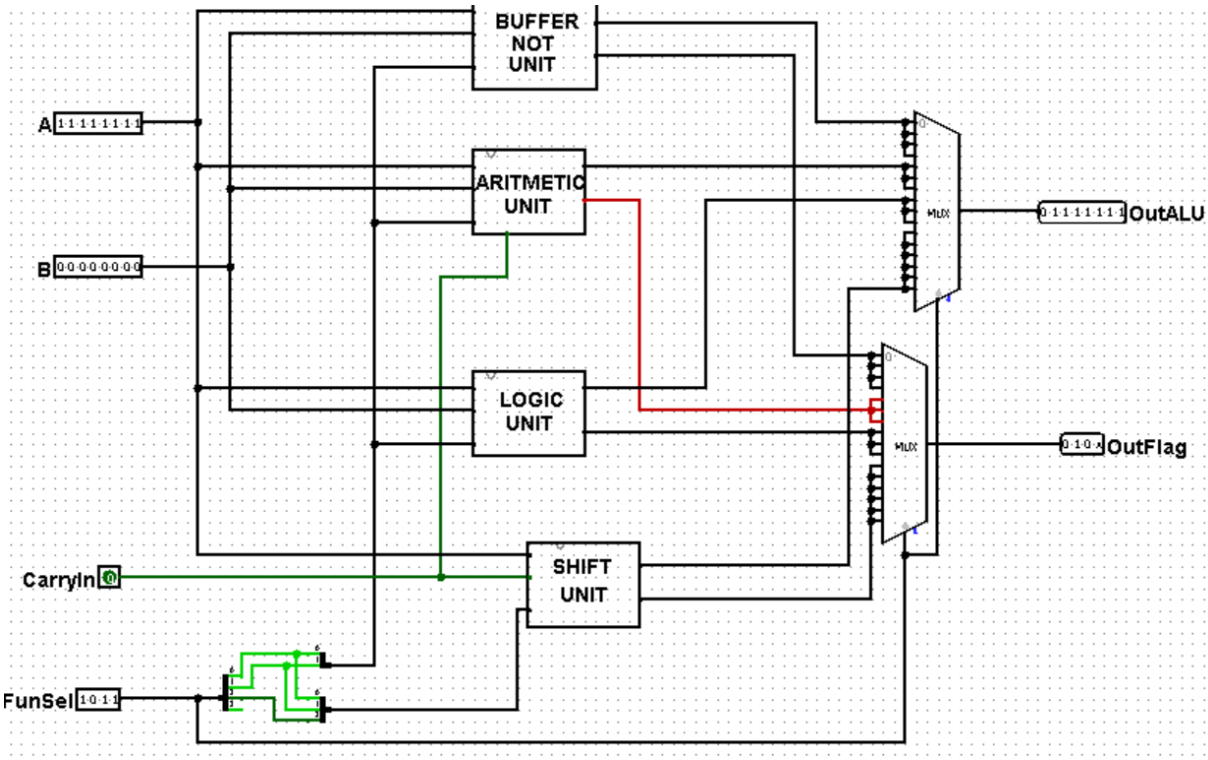

Figure 21: ALU file in the Logisim Circuit while applying first test case datas to design

- RegSel of Address Register File: 000
- FunSel of Address Register File: 00
- OutCSel of Address Register File: 00
- OutDSel of Address Register File: 00
- MuxASel: 00
- MuxBSel: 00

- And if we enter 1 full clock cycle, we can see our R1 = 0xFF, R2 = 0x7F, N = 0, C = 1 in the following Figure 22.

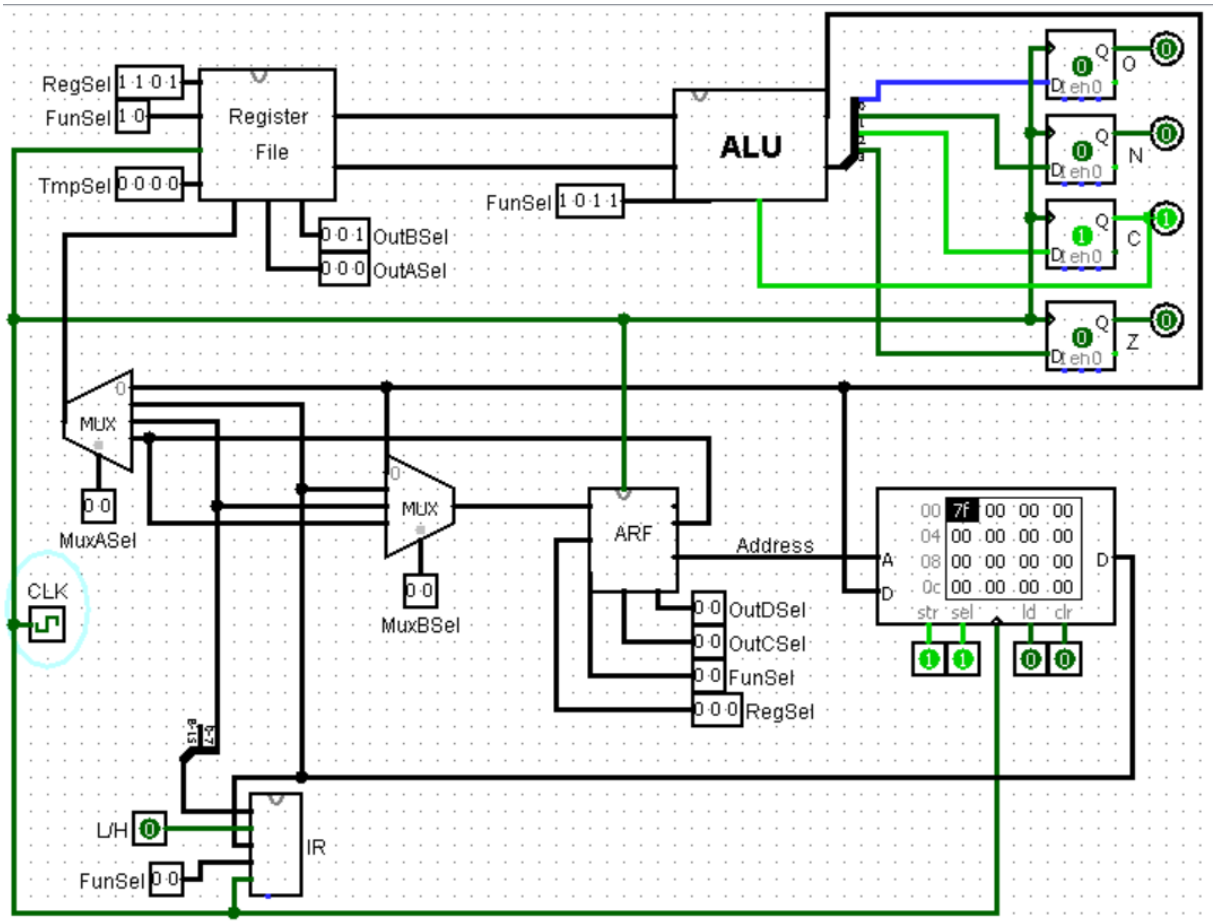

Figure 22: Logisim Circuit while applying second test case datas to design

- And if we enter the Register file during the simulation, we can see our Register2 output = 0x7F, and in the Register1 output = 0xFF following Figure 23.

- For the 3rd test case, our aim is to Load SP with the value of R3 value. Our input values will be in the following manner:

  - RegSel of Register File: 1101

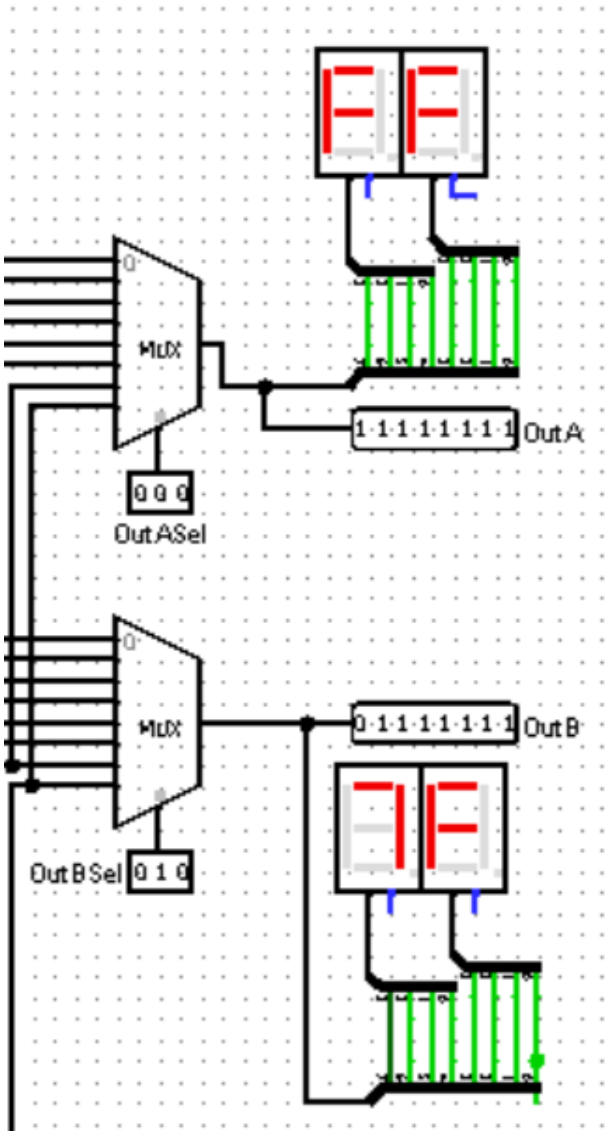

Figure 23: Register File in the Logisim Circuit while applying second test case datas to design

- TmpSel of Register File: 0000
- FunSel of Register File: 10
- OutASel of Register File: 010
- OutBSel of Register File: 001
- FunSel of ALU: 0000
- RegSel of Address Register File: 110
- FunSel of Address Register File: 10
- OutCSel of Address Register File: 00
- OutDSel of Address Register File: 00
- MuxASel: 00
- MuxBSel: 00

- And if we enter 1 full clock cycle, we can see our R1 = 0xFF, R3 = 0x7F, SP=0x7F, N = 1, C = 1. See the SP value in the Address register file during simulation in the following Figure 24.

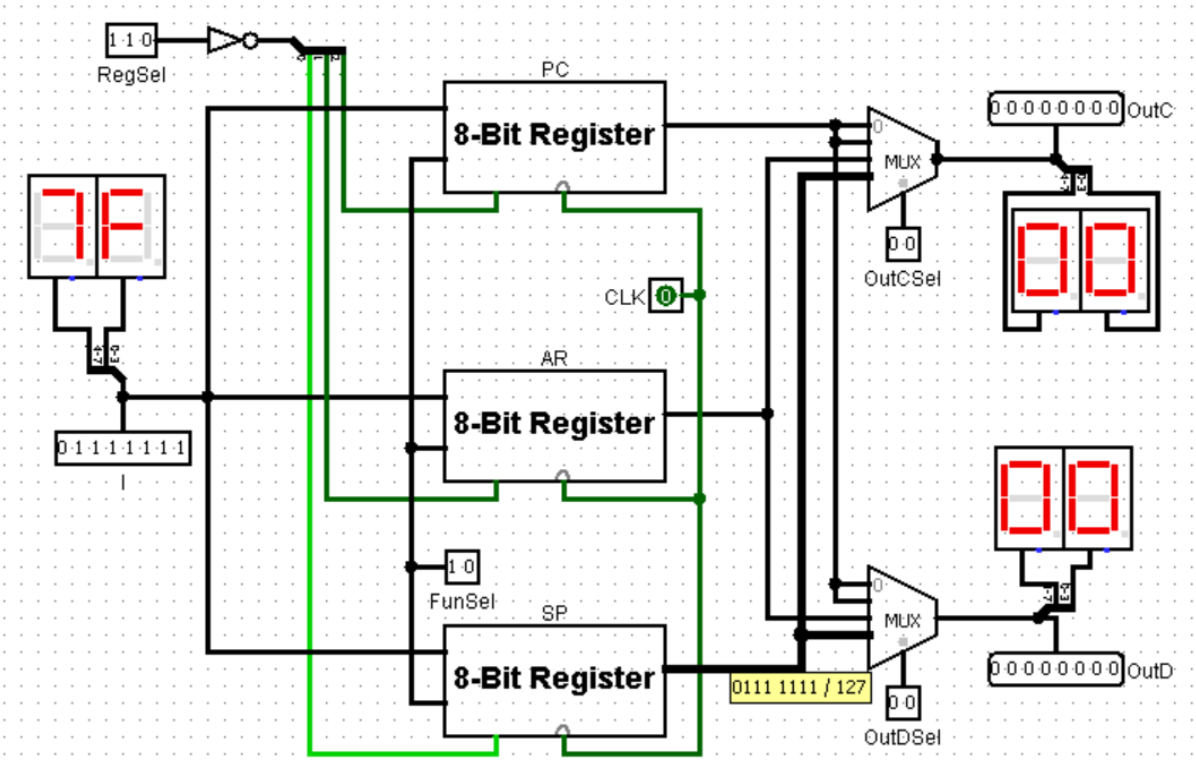

Figure 24: SP value of Logisim Circuit while applying third test case datas to design

- And if we enter the Register file during the simulation, we can see our Register2 output = 0x7F, and in the Register1 output = FF following Figure 25.

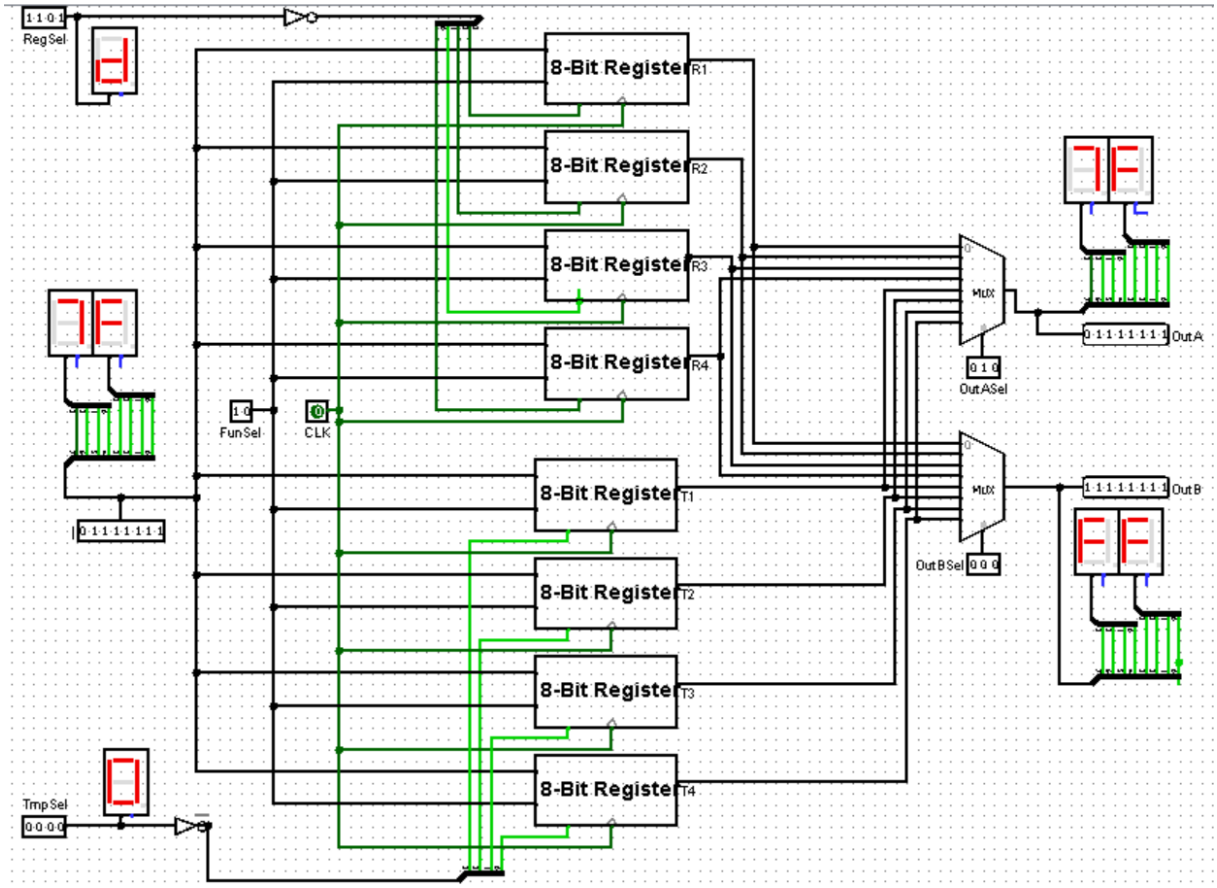

Figure 25: Register File in the Logisim Circuit while applying third test case datas to design

4 DISCUSSION

For the Part 1, we are asked to implement 2 different types of registers: 8-bit register and 16-bit register. Firstly, we implement a 8 bit register, that process its input values and operates accordingly as given in the table in the assignment documentation. Therefore, we implement our circuit with D flip flops, adders, subtracters and multiplexer. Then, we implement a 16 bit IR register, that process its input values and operates accordingly as given in the table in the assignment documentation. To do so, we implement our circuit with D flip flops, adders, subtracters, demultiplexer and multiplexer. By that our circuit process its inputs and behaves as it is given in the tables.

In the second part, we are asked to implement a register file that works with the constraints given in the assignment document. Firstly, we implement a system shown which consists of four 8-bit general purpose registers and four 8-bit temporary registers, and other details given in the document. This circuit process its inputs and operates according to the tables given. for example for some inputs only certain registers are enables, or our value is incremented or decremented etc.. Then, we implement a address register file (ARF) system which consists of three 8-bit address registers: program counter (PC), address register (AR), and stack pointer (SP). This circuit takes the inputs OutCSel OutDSel, FunSel, OutC, OutD, RegSel, process them, and according to the tables in the document, it directs the input to the different register parts.

For the third part, we are asked to implement an Arithmetic Logic Unit (ALU) that has two 8-bit inputs, an 8-bit output, and a 4-bit output for zero, negative, carry, and overflow flags. This ALU behaves according to the tables in the corresponding part of the documentation. It takes A and B data, FunSel and a Cin, and yield OutALU, OutFlag values. This ALU is able to make many operations such as, Circular, Arithmetic, Logical Left or Right Shifts, and yield appropriate flags for the operation results.

And lastly for the part, we implement a organized circuit consists of the parts that we have implemented before for this assignment. In this circuit, all of it uses the same clock signal. We connected the circuits as given in the circuit picture in the assignment documentation. We connected register file to ALU and ALU to Memory, Address Register via a MUX, and Register File via MUx, and we connected IR to these MUXs. With this circuit, we used all of our circuits together. Then we tested this organized circuit with the given test cases to verify our implementations.

5 CONCLUSION

In conclusion, in this assignment we implemented different types of registers, register files, address register file, ALU, and a final organized circuit using all of these circuit. It was not really hard to figure out how to construct our circuits and their logics, since we are already experienced from Digital Circuits Course and Computer Organization Course. We practiced creating circuits using Logisim . Again we have seen that Logisim is very useful for implementing the circuits. We were able to get diagrams and outputs of the circuits easily, we have written in Logisim.

In the Part1 we have implemented 2 different register circuits to use later, that takes necessary inputs, operates accordingly and gives output values, according to the information given in the documentation.

Then for the Part 2, we easily implemented the register file and address register file, that takes necessary inputs, operates accordingly and gives output values. In this part having 8 different register members made the question a little difficult but we implemented it successfully.

Then for the Part 3, we implemented a ALU that takes FunSel, and yields some outputs and flags according to the critierias given in the documentation. This ALU is able to make Circular or Arithmetic or Logical Shifts to Left, Right as in the pictures.

For the Part 4 we had not faced difficulties due to the fact that we have already constructed the neccessary circuits and the only thing we need to do is wire them according to the circuit picture in the document, and tested with the test cases which are given in the Ninova by our Teaching Assistant. In this assignment creating circuits was not hard but understanding their logic made us think and took our time a bit. After all, we practiced our knowledge on the LaTex language and Logisim program. This experiment was harder than any of the previous ones but as we are getting more comfortable while using programs and languages and working as a team. So we concluded our work successfully.