ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO	:	1
DUE DATE	:	19.05.2021
GROUP NO	:	G8

GROUP MEMBERS:

150190014	:	FEYZA ÖZEN (Group Representative)
150180086	:	EFE YİĞİT TAŞ
150170050	:	MERT YEREKAPAN

1 INTRODUCTION

In this project, we designed registers and register files. Then we implemented different types of circuits by using them as libraries. We designed an IR register, a register file, an address register file, an ALU, and the combination of them. Each part has been used as libraries later.

2 PROJECT PARTS

2.1 PART 1

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

2.1.1 8-Bit Register

The 8-bit register that we designed in Part 1.a has 4 functions. These functions are controlled by 2-bit FunSel input and an Enable input. Figure 1 shows the characteristics of 8-bit register.

E	FunSel	Q^+	
0	ϕ	Q	(Retain Value)
1	00	Q-1	(Decrement)
1	01	Q+1	(Increment)
1	10	I	(Load)
1	11	0	(Clear)

Figure 1: Characteristic Table for Part 1.a

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

- **Multiplexer**: There are 4 different functionalities in this part. We used a 4:1 multiplexer to select and operate these functions. Figure 1 was our reference for deciding inputs of multiplexer.
- **D FLIP FLOP**: This is a library that we designed to minimize the circuit in the Figure 2. As shown in Figure 3, it has 8 parallel D Flip Flops. Its 8-bit input is the output of MUX. D Flip Flops are used as the memory units.
- **ADDER**: It is 8-bit full adder. We used it to operate the third operation in the table in Figure 1. ADDER's first input is the 8-bit output of D FLIP FLOP, second input is constant 1 value. The output goes to the second input of MUX.

- **SUBTRACTOR**: It is 8-bit Subtractor. We used it to operate the second operation in the table in Figure 1. SUBTRACTOR's first input is the 8-bit output of D FLIP FLOP, second input is constant 1 value. The output goes to the first input of MUX.

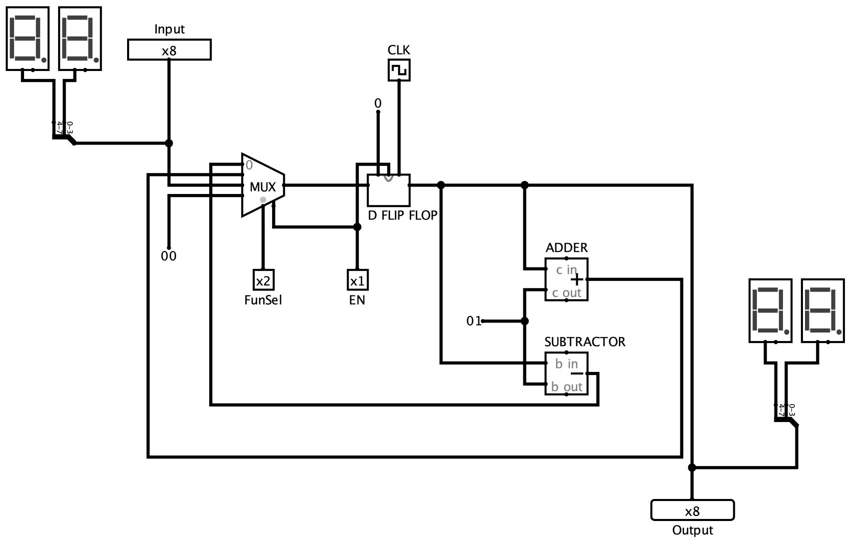


Figure 2: Logisim Circuit For Part 1.a

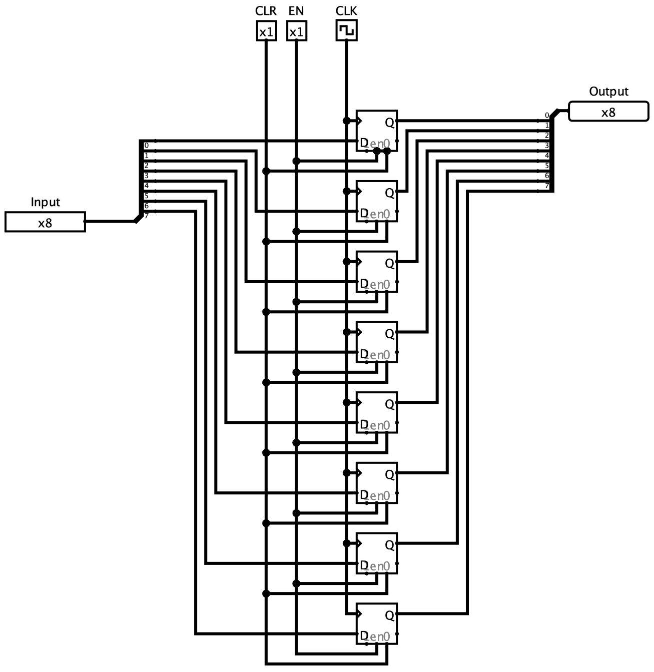


Figure 3: Logisim Circuit For D FLIP FLOP(8-bit)

2.1.2 16-Bit Register

The 16-bit IR register that we designed in Part 1.b has 5 functions. These functions are controlled by 2-bit FunSel input, L/H input, and an Enable input. Figure 4 shows the characteristics of 16-bit IR register.

$L/\overline{H}$	Enable	FunSel	IR^+	
ϕ	0	ϕϕ	IR	(Retain Value)
ϕ	1	00	IR - 1	(Decrement)
ϕ	1	01	IR + 1	(Increment)
0	1	10	IR(8-15) <- I	(Load MSB)
1	1	10	IR(0-7) <- I	(Load LSB)
ϕ	1	11	0	(Clear)

Figure 4: Characteristic Table for Part 1.b

As shown in Figure 5, there are following components in our design:

- **Multiplexer (4:1)**: There are 5 different functionalities in this part. We used a 4:1 multiplexer to select and operate these functions. Figure 4 was our reference for deciding inputs of multiplexer. 00, 01, and 11 conditions are the same as in Part 1.a. For 10 condition, we used another multiplexer to operate the fourth and fifth operation in the table in Figure 4

- **Multiplexer (2:1)**: We used this multiplexer to operate the fourth and fifth operation in the table in Figure 4. Its selection input is L/H. Its inputs are the combination of selected bits of the input and other bits of the output. It allows us to load only MSB or LSB to the register.

- **D FLIP FLOP**: This is a library that we designed to minimize the circuit in the Figure 5. As shown in Figure 6, it has 16 parallel D Flip Flops. Its 16-bit input is the output of MUX(4:1). D Flip Flops are used as the memory units.

- **ADDER**: It is 16-bit full adder. We used it to operate the third operation in the table in Figure 1. ADDER's first input is the 16-bit output of D FLIP FLOP, second input is constant 1 value. The output goes to the second input of MUX(4:1).

- **SUBTRACTOR**: It is 16-bit Subtractor. We used it to operate the second operation in the table in Figure 1. SUBTRACTOR's first input is the 16-bit output of D FLIP FLOP, second input is constant 1 value. The output goes to the first input of MUX(4:1).

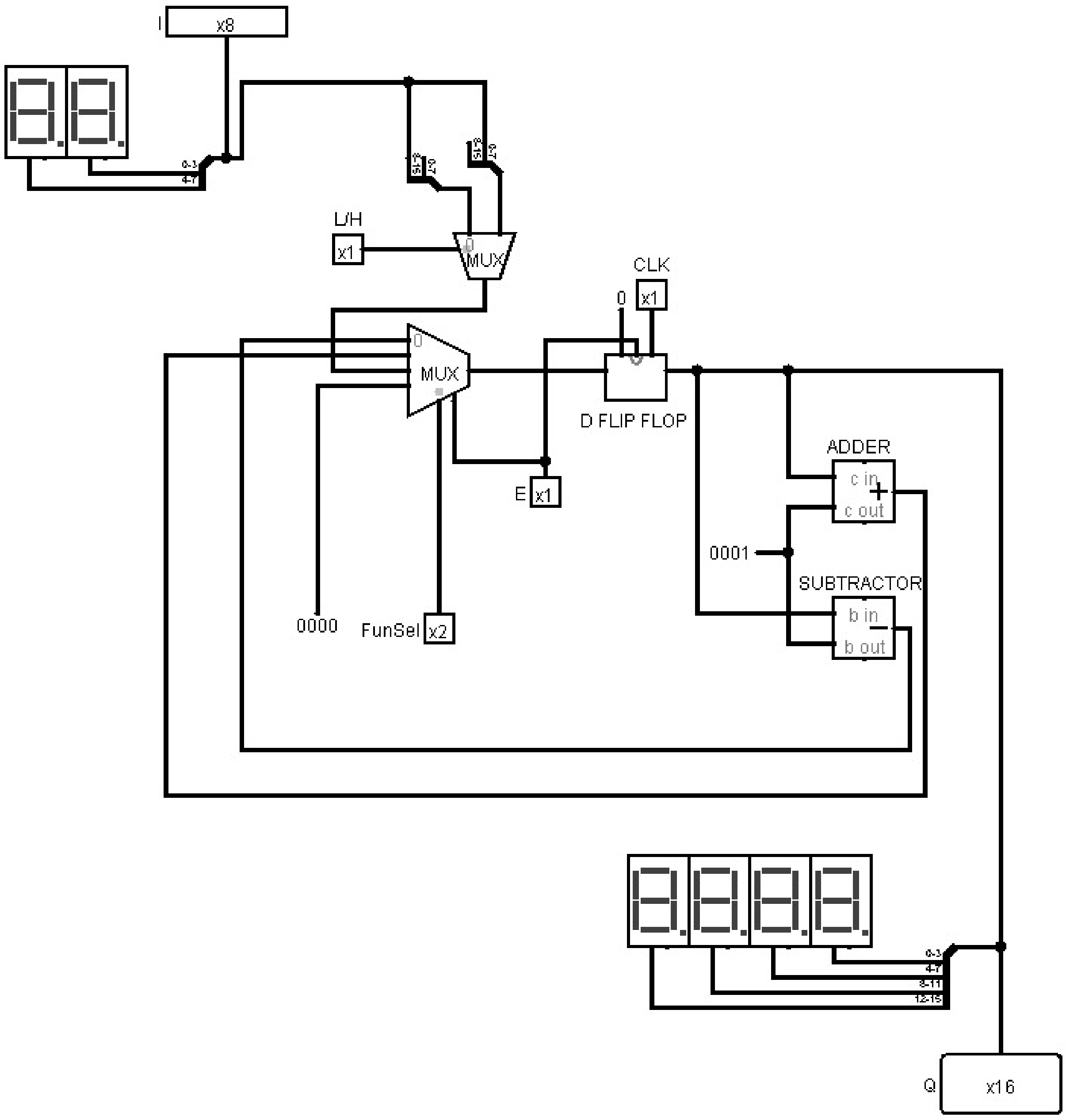


Figure 5: Logisim Circuit for Part 1.b

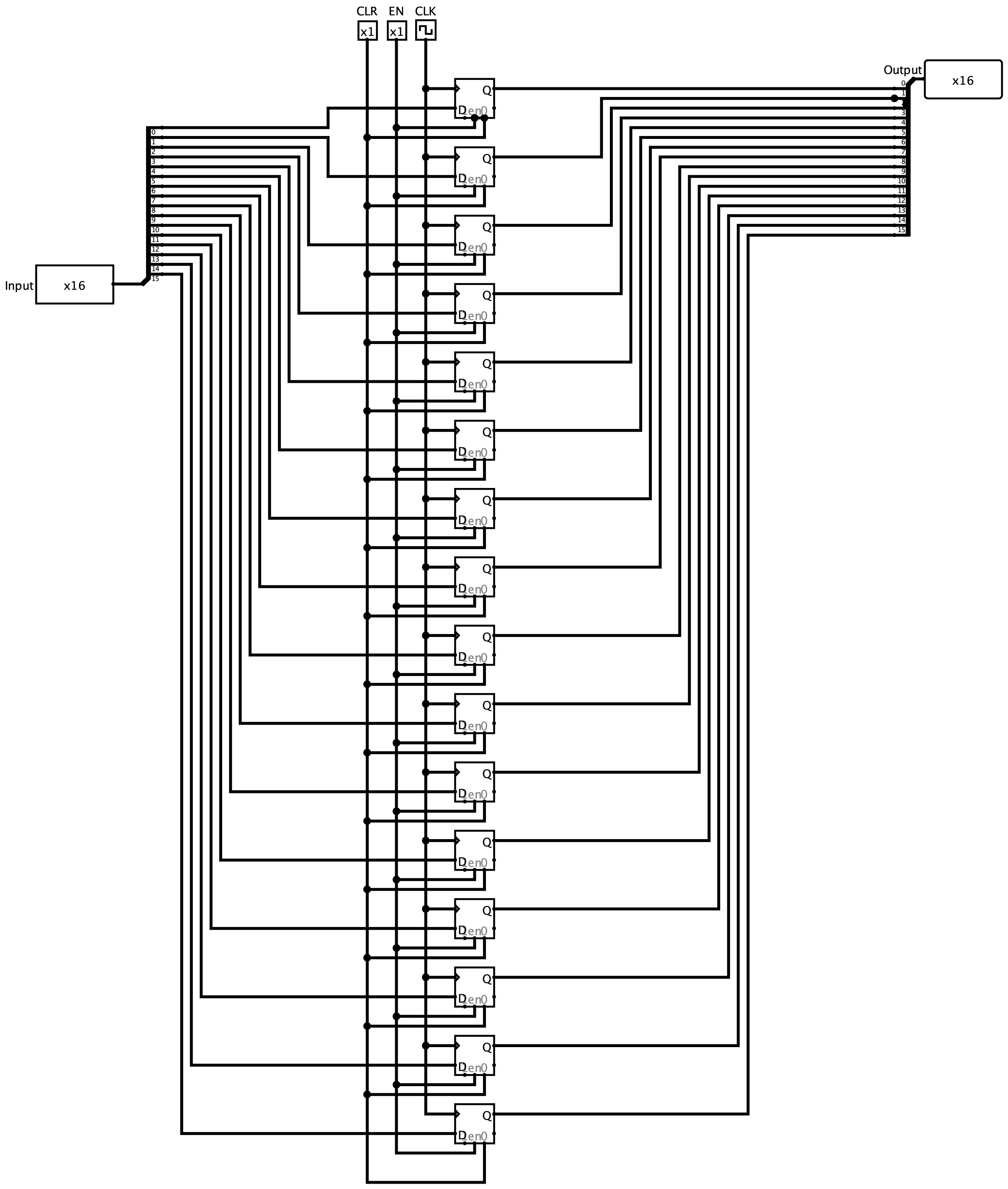


Figure 6: Logisim Circuit for D FLIP FLOP(16-bit)

2.2 PART 2

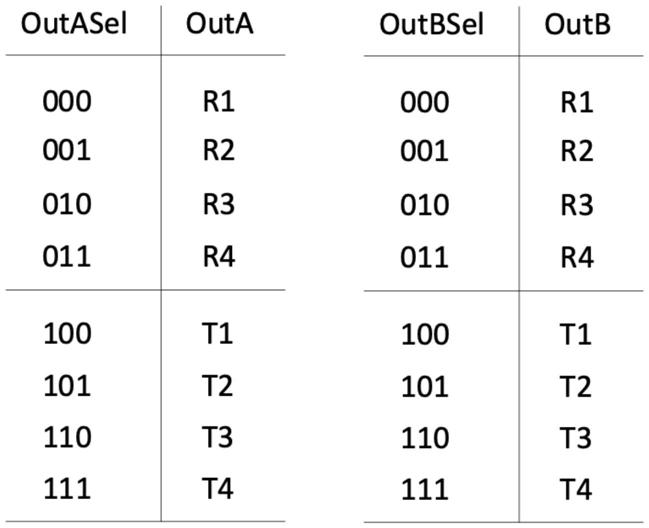
2.2.1 Register File

In this part we designed a register file. It consists of four 8-bit general purpose registers: R1, R2, R3, and R4 and four 8-bit temporary registers: T1, T2, T3, and T4. Figure 7 shows the characteristics of OutASel and OutBSel inputs.

As shown in Figure 8, there are following components in our design:

- **8-Bit Register**: We used 8 8-Bit Register, 4 of them are general-purpose registers. The remaining ones are temporary registers. The registers that we used are the library of the circuit in Part 1.a. RegSel and TempSel inputs are selecting which register is going to be enabled.

- **Multiplexer**: We used 2 multiplexers to select which register's output is going to be selected. Their inputs are OutA and OutB. Figure 7 was our reference for deciding inputs of multiplexer.



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 7: Characteristic Table for Part 2.a

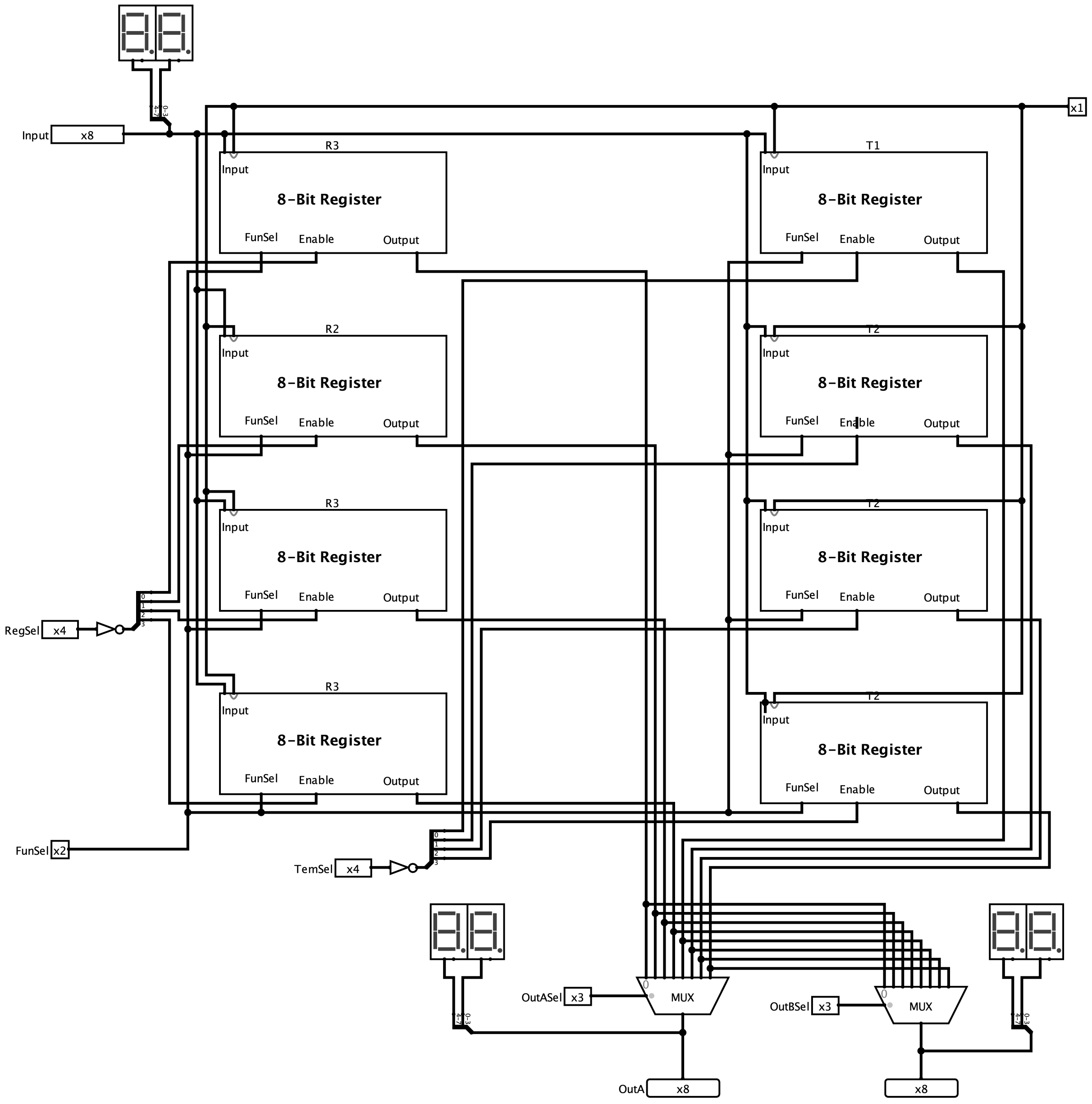


Figure 8: Logisim Circuit For Part 2.a

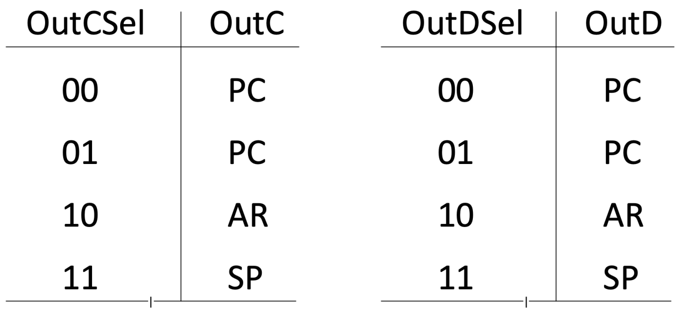
2.2.2 Address Register File

In this part, we designed an address register file (ARF) system which consists of three 8-bit address registers: program counter (PC), address register (AR), and stack pointer (SP).

As shown in Figure 10, there are following components in our design:

- **8-Bit Register**: We used 3 8-Bit Register. These are a program counter (PC), an address register (AR), and a stack pointer (SP). The registers that we used are the library of the circuit in Part 1.a. RegSel input selects which register is going to be enabled. FunSel works as in Part 2a.

- **Multiplexer**: We used 2 multiplexers to select which register's output is going to be selected. Their inputs are OutCSel and OutDSel. Figure 9 was our reference for deciding inputs of multiplexer.



OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

OutDSel	OutD
00	PC
01	PC
10	AR
11	SP

Figure 9: Characteristic Table for Part 2.b

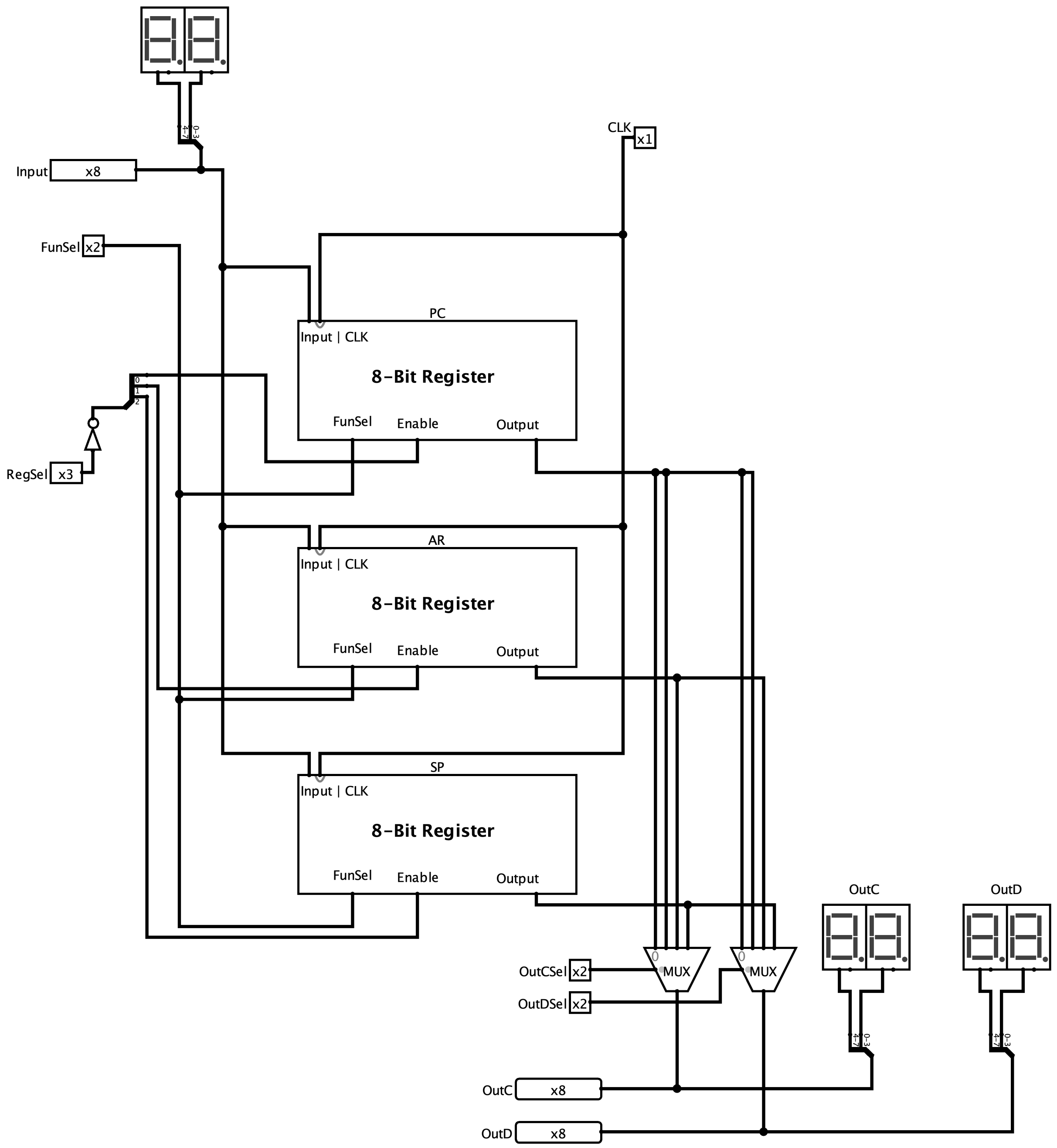


Figure 10: Logisim Circuit For Part 2.b

2.3 PART 3

In this part, we designed 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. Figure 11 shows the characteristics of ALU.

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

- **Multiplexer**: We used a 16:1 multiplexer to select which arithmetic operation is going to be selected. Its select input is 4-bit FunSel. Figure 9 was our reference for deciding inputs of multiplexer.

- **Arithmetic Unit**: This is a library that we designed to minimize the circuit in Figure 12. The circuit of it is shown in Figure 13. This circuit can operate 3 of operations. These operations are A + B, A + B + Carry, and A - B. To do them it has a full bit adder inside. We used a 4:1 multiplexer to select operations. Its select input is first two bits of FunSel. The second bit of FunSel chooses the operation between adding and subtracting. To do the selection we used an XOR gate. It allows us to apply 2's complement method. This Arithmetic Unit gives Carry Out, Overflow output according to the operation.

- **Shift Unit**: This is a library that we designed to minimize the circuit in Figure 12. The circuit of it is shown in Figure 17. We used 3 8:1 multiplexers to select operation, carry, and overflow. Their select inputs are the same and first 3 bits of FunSel. We used splitters to split bits and then shift them.

- **Zero Checker**: This is a library that we designed to minimize the circuit in Figure 12. The circuit of it is shown in Figure 15. We used a zero checker and some splitters to get ZCNO output.

FunSel	OutALU	Z	C	N	O
0000	A	√	–	√	–
0001	B	√	–	√	–
0010	NOT A	√	–	√	–
0011	NOT B	√	–	√	–
0100	A + B	√	√	√	√
0101	A + B + Carry	√	√	√	√
0110	A - B	√	√	√	√
0111	A AND B	√	–	√	–
1000	A OR B	√	–	√	–
1001	A XOR B	√	–	√	–
1010	LSL A	√	√	√	–
1011	LSR A	√	√	√	–
1100	ASL A	√	–	√	√
1101	ASR A	√	–	–	–
1110	CSL A	√	√	√	–
1111	CSR A	√	√	√	–

Figure 11: Characteristic Table for Part 3

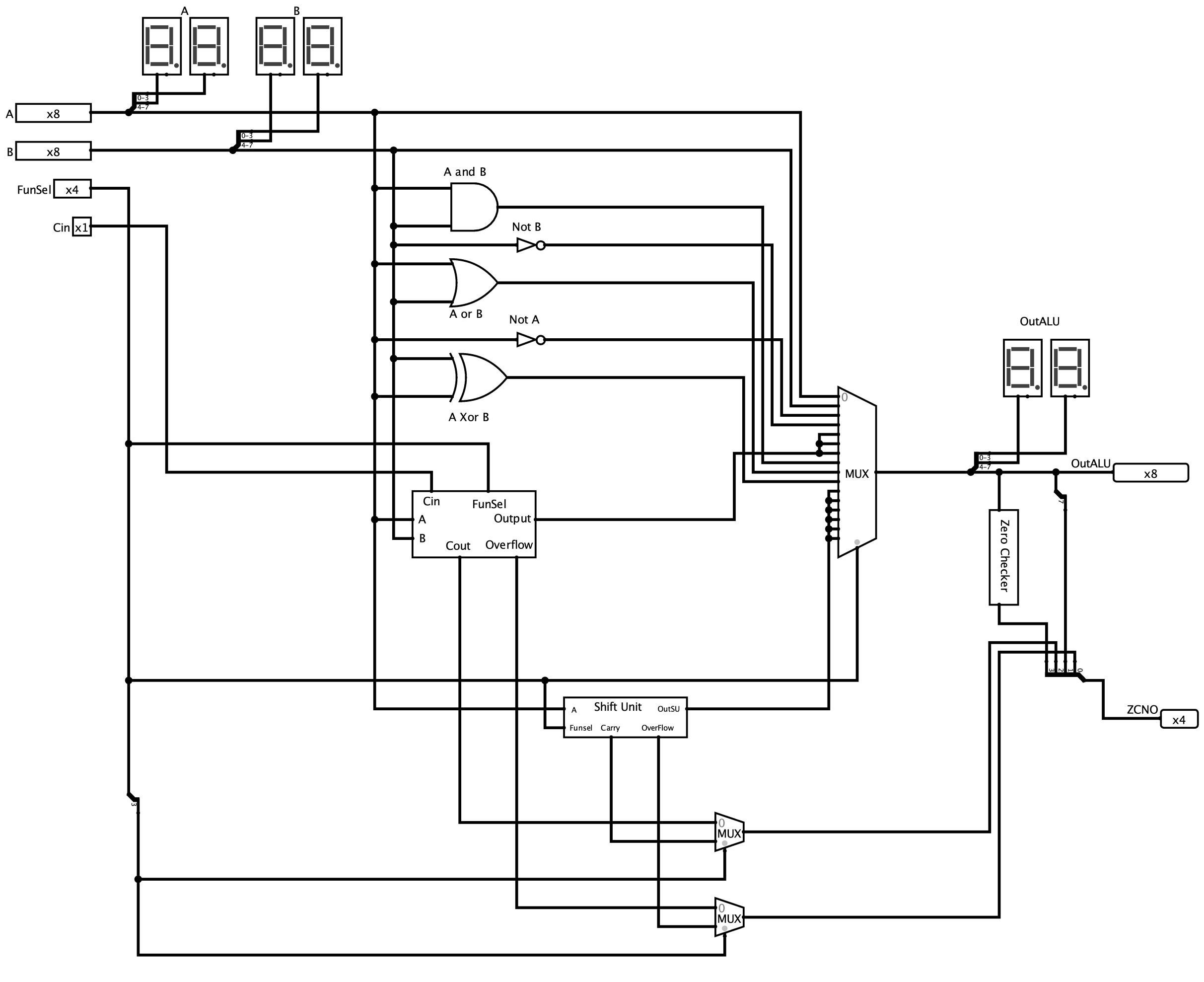


Figure 12: Logisim Circuit For Part 3

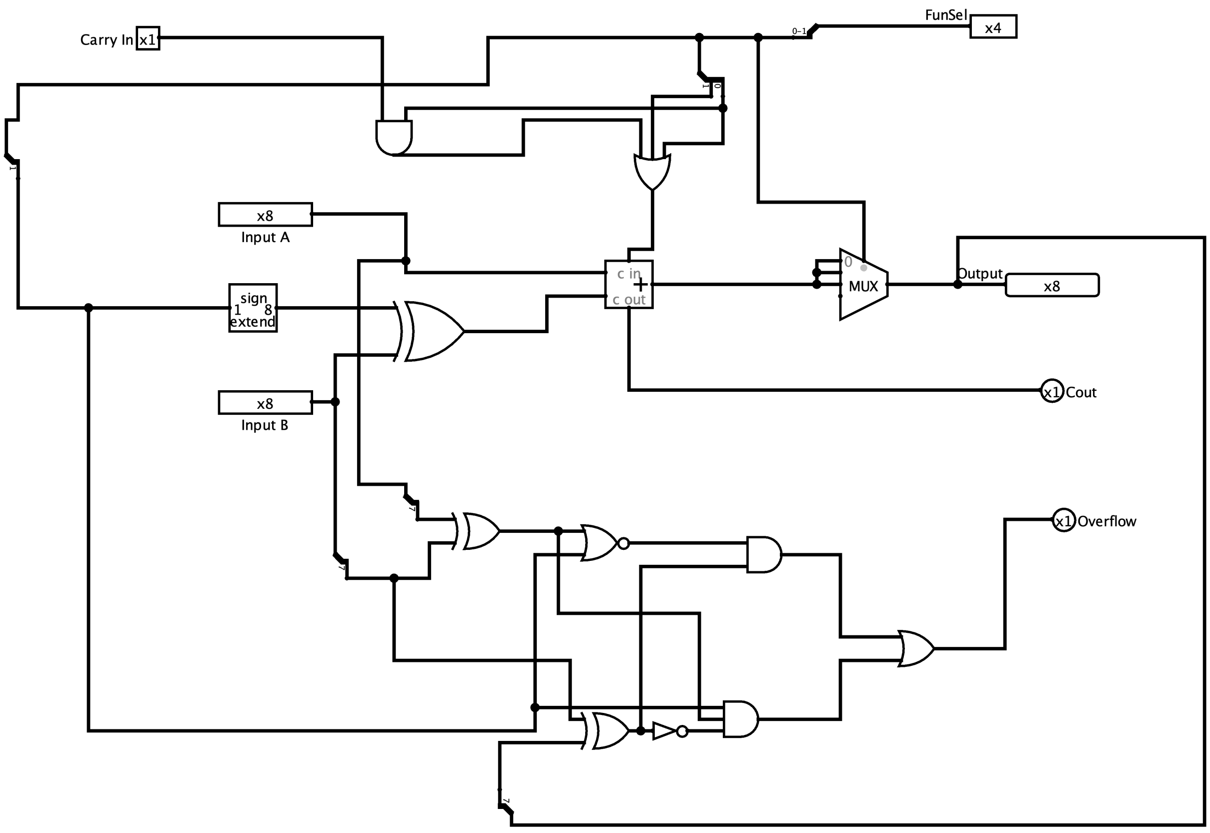


Figure 13: Logisim Circuit For Arithmetic Unit

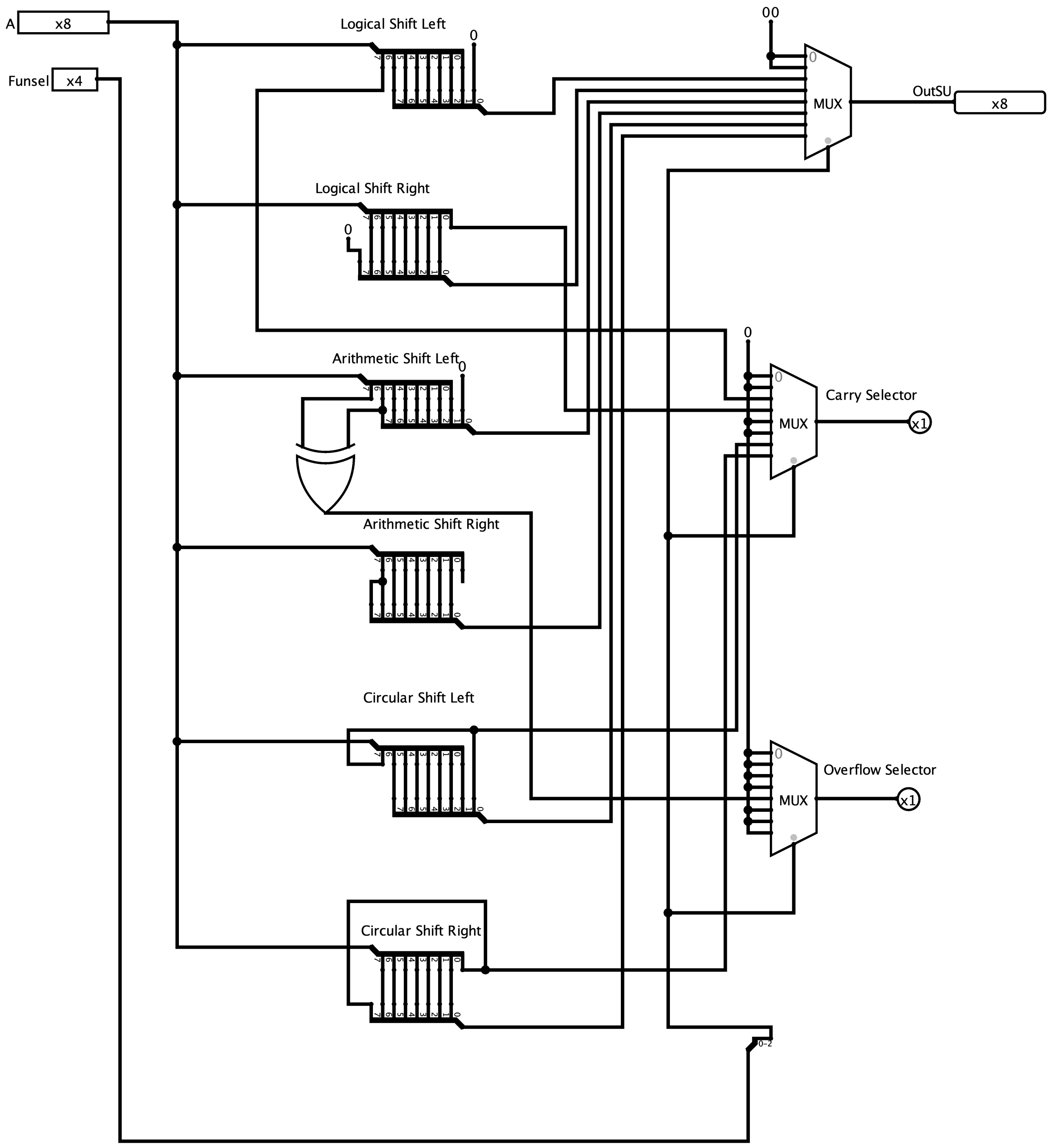


Figure 14: Logisim Circuit For Shift Unit

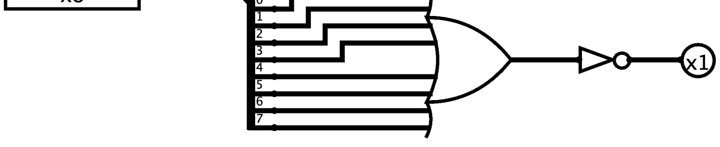


Figure 15: Logisim Circuit For Zero Checker

2.4 PART 4

ZCNO Register: We use this to store the values from ZCNO output of ALU. Z is for Zero, C for Carry, N for Negative and O for Overflow. We used register to store them.

We did not implement additional libraries in this part. Therefore we explained Part 4 more in detail in the discussion part.

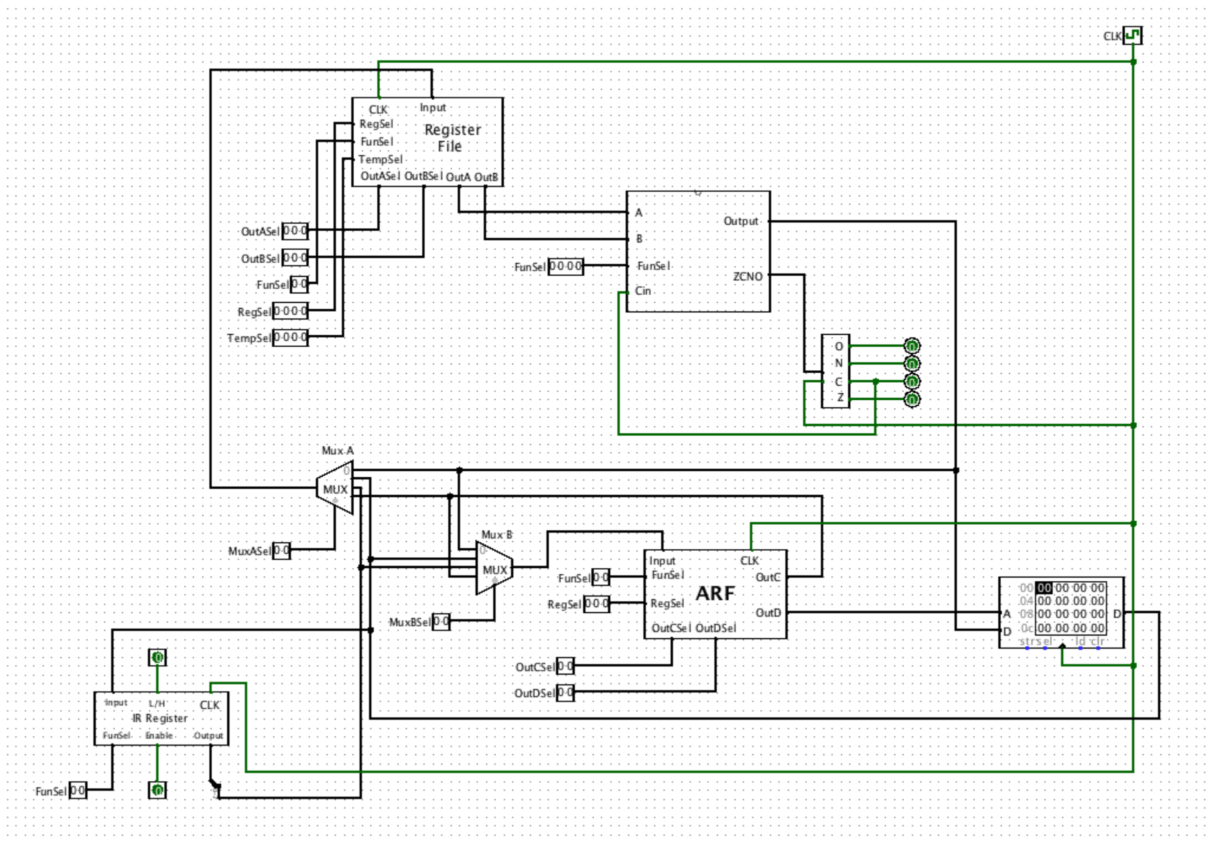


Figure 16: Final System

3 RESULTS

- Since we want to load R1 with NOT (R2), we want to select outB in the register as R2, so OutBSel (0000 0000) should be 001. Our ALU operation should be FunSel 0011 since it should be NOT B (1111 1111). The value from ALU goes to memory and MuxA. MuxASel must be 00 and subtract OutALU (1111 1111). Then OutALU becomes the input for the Register File. The FunSel of the register becomes 10 and it is loaded. The value from ALU goes to memory and MuxA. MuxASel must be 00 and subtract OutALU. Then OutALU becomes the input for the register. The FunSel of the register becomes 10, it becomes clear that the load will be done, and then the RegSel becomes 0111 and our value (1111 1111) loads into R1.

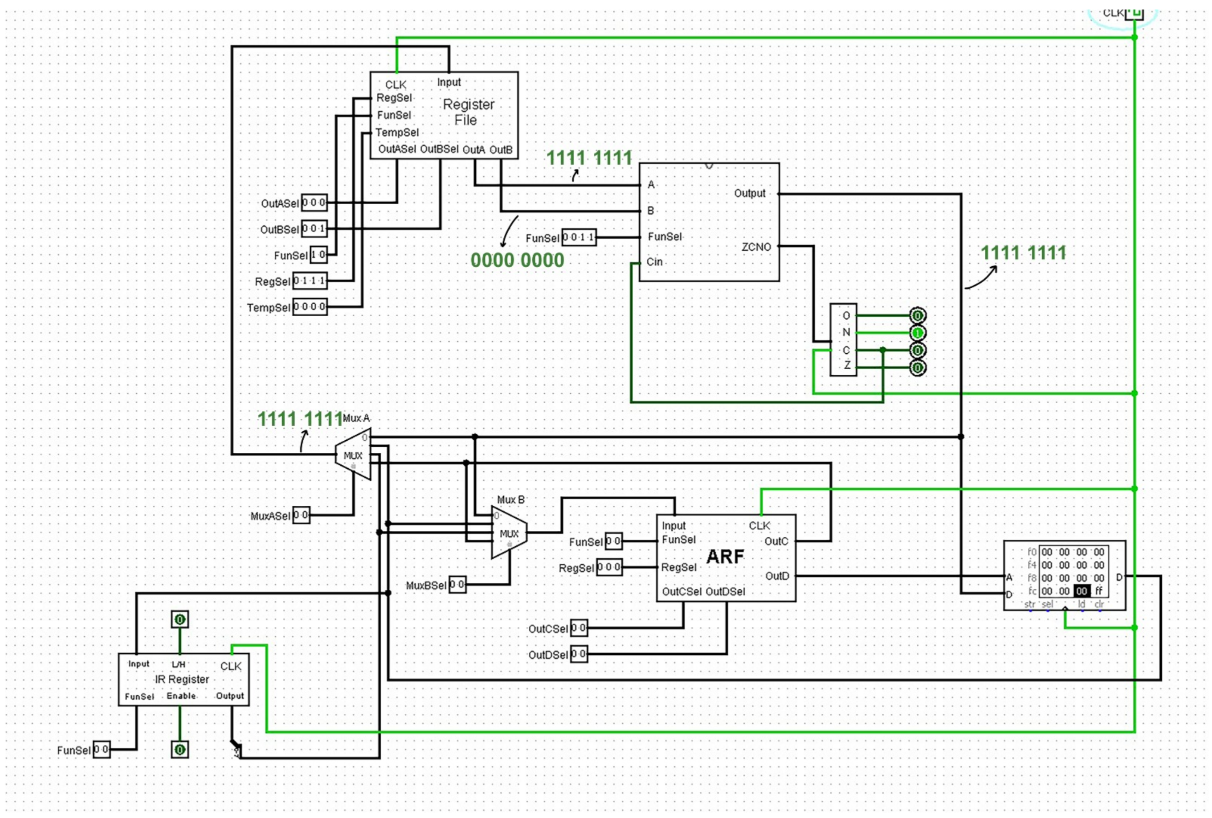


Figure 17: Test Case 1

- We want to Load R3 with LSR(R1), we want to select outA in the register as R1, so outASel should be 000 so A input of ALU will be 0xFF. FunSel of ALU will be 1011 so we can perform LSR on A input. Output of ALU wil go to MuxASel which is set 00 so it selects the ALU output and gives to Register File as input. We will Load R3 with value of LSR(R1), so we set RegSel of register file 1101 and Funsel of Register File as 10 (Load operation)

- We want to load SP with the value of R3, we will set OutASel of register file as 010 (R3 is selected) so A input of ALU will be 0x7F. We will select FunSel of ALU as 0000 so it will stay same. Output of ALU will go to MuxB and we will select MuxBSel as 00 so MuxB outputs OutALU. MuxB output is the input of Address Register File. Regsel of ARF is 110 to enable SP and Funsel of ARF is 10 which is LOAD operation. After the clock is enabled SP will contain the R3 value which is 0x7F.

4 DISCUSSION

This project is good for understanding and applying registers and register files. We learned how to use registers for storing data and how can we get different circuits with various functionalities.

We used the software Logism a lot through the process. That makes us more familiar with the program. We learned new features of Logism such as libraries. Libraries are quite useful and make the process much easier.

We designed each part of this project as a different library. For this part we need to put everything together and create a system that makes sense and work. To do this, we imported the libraries we created in the previous parts.

System works in one clock cycle and uses all the parts we implemented. Logic of this system works as follows:

We have a Register File, implemented in Part2a, allows us to select which registers to use with OutASel and OutBSel inputs, allows us to select which operation we will perform in that register such as whether decrementing, incrementing, loading or clearing with Funsel input. We use Register File to select which register will be selected and what will they do for this clock cycle. Outputs goes to ALU.

Register File is connected to ALU, implemented in Part3, allows us to perform numerous actions on inputs OutA and OutB. We take this inputs from RegisterFile as output and connect to ALU. Funsel input of ALU determines the operation we will perform on the inputs. Output of ALU goes to memory and MuxA, depending on the MuxASel, it is connected to Register File again. We also have a 4 bit Flag Register shows 4 attributes depending on the operation such as Zero, Carry, Negative and Overflow.

We also have Address Register File and it consists of 3 8-bit address registers: PC, AR and SP. It takes an 8-bit input from MuxBSel, OutCSel and OutDsel to select which address registers will be selected from PC, AR and SP. Funsel allows us to select which operation we will perform in those registers such as whether decrementing, incrementing, loading or clearing. We will use this to get an input from MuxB weather OutALU, Memonery Output, IROut(0-7) or ARF OutC, then select which operation to perform and later giving the address as an output to the Memory.

Another part of this homework is IR which is Instruction Register which takes 8-bit input from Memory's output, a L/H to determine which part to use, enable and funsel. Its output goes to MuxA and MuxB.

5 CONCLUSION

We had hard times to understand the logic of Instruction Register at first. In the last part it was also hard to imagine what memory does, how it works. Besides that it was pretty clear what to do from the homework file. We designed a basic system selects registers from many registers, make operations with ALU and manages memory with

Address Register File.