ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION REPORT

PROJECT NO : 2

SUBMISSION DATE : 03.04.2019

GROUP NO : 12

GROUP MEMBERS:

150160151 : Esin Ece AYDIN

150170016 : Umit BAŞAK

150180704 : Cihat AKKİRAZ

150180705 : Batuhan Faik DERİNBAY

150180707 : Fatih ALTINPINAR

SPRING 2019

1 INTRODUCTION

In the era of modern computing, all the processing units go through 3 stages. Fetch, execute and decode. On the fetch stage, the next instruction is fetched from the memory. Then, the central processing unit (CPU) will need to understand fetched instruction on the decode stage. Finally, CPU can execute operations based on the decoded information. Arithmetic Logic Unit (ALU) performs needed operations on the executing stage. ALU, is one of the most important parts of a CPU. The combinational digital electronic circuit (referring to ALU) designed with the aim of doing arithmetic, bitwise logical and bit shift operations on binary numbers.

In this project, an ALU is designed and implemented that has two 8-bit inputs and an 8-bit output. Then basic CPU architecture is designed via using designed register files from the previous project.

2 METHODOLOGY

2.1 Part-1

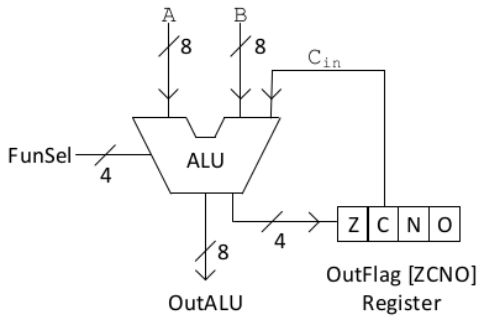

Figure 1: Arithmetic Logic Unit(ALU) Structure

In the ALU, there are two binary 8-bit inputs, a 4-bit FunSel input to select operation that will be processed and carry input, a 4-bit OutFlag output to display whether the flags given below are active, an 8-bit OutALU to display the result of the operation.

- **Z(zero)** bit is set if **OutALU** is zero.
- **C(carry)** bit is set if **OutALU** sets the carry.
- **N(negative)** bit is set if the ALU operation generates a negative result.
- **O(overflow)** bit is set if an overflow occurs.

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

Table 1: Characteristic table of Arithmetic Logic Unit (ALU)

Arithmetic Logic Unit that is designed in this project consists three parts: Arithmetic unit, logic unit and shifting unit (see Figure 2). Performed operation is declared via 4-bit FunSel signal. FunSel signal is connected to 16:1 multiplexer's selector line and the output of the multiplexer is coming up based on FunSel signal. To process logical operations (ranging between $(0000-0011)_2$ and between $(1000-1001)_2$ in Table 1) the logic unit, to process arithmetic operations (ranging between $(0100-0111)_2$) the arithmetic unit and to process shifting operations (ranging $(1010-1111)_2$) the shifting unit are designed.

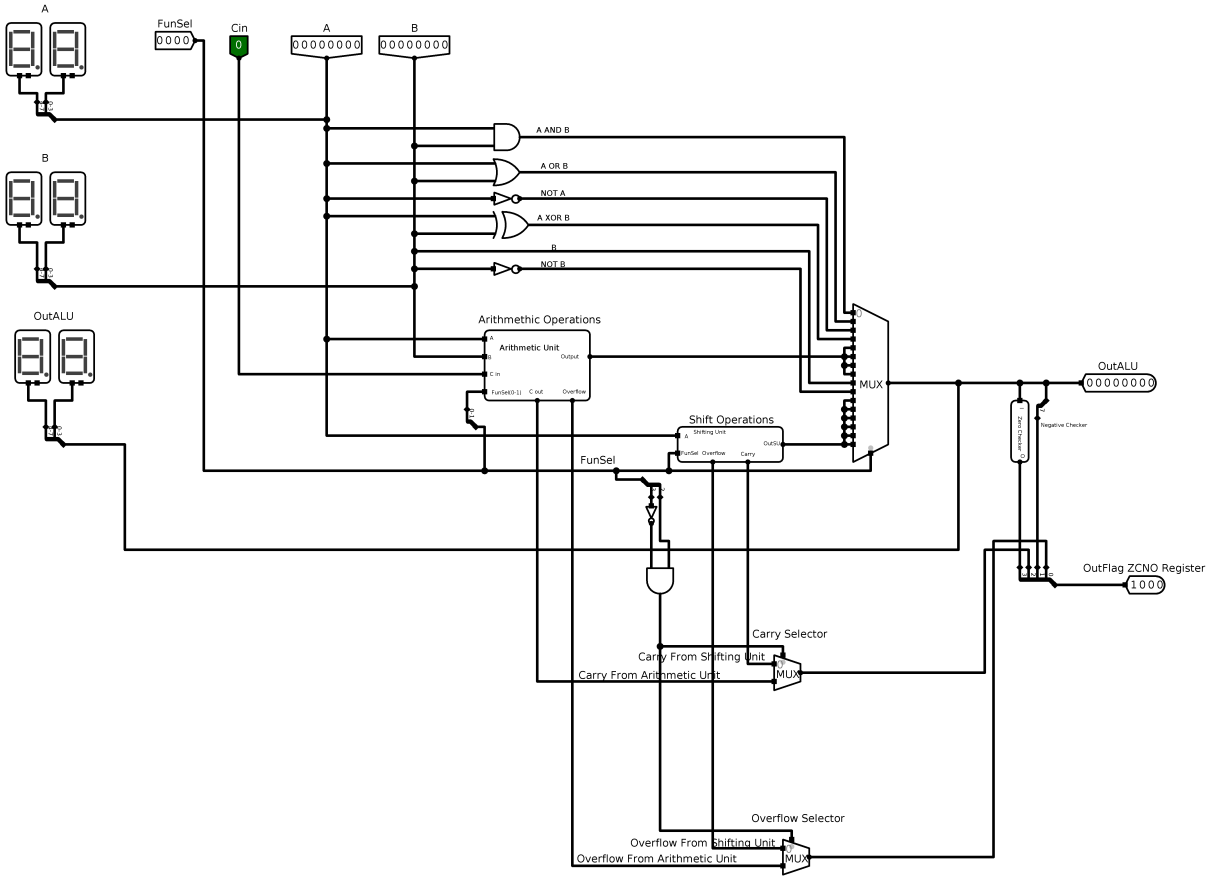

Figure 2: Arithmetic Logic Unit

In order to determine whether the OutALU is zero, basic ZeroChecker circuit is designed as in Figure 3. When the output of this circuit is 1, Z bit will be 1 on the OutFlag.

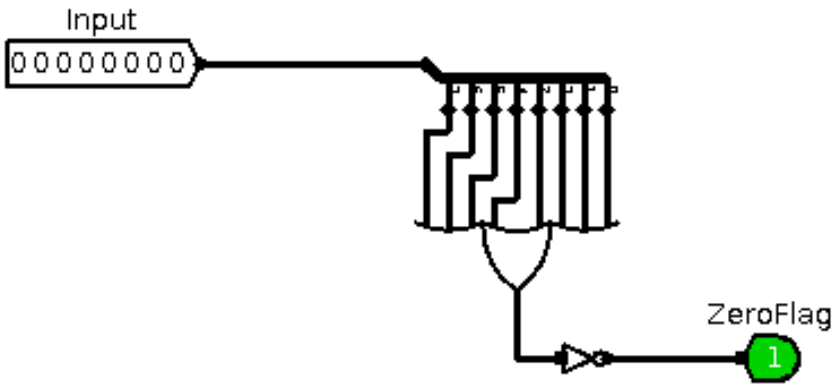

Figure 3: Zero Checker Circuit

2.1.1 Part-1a

In this section, in order to implement the function selection (FunSel) signal 4 through 7 or $(0100 - 0111)_2$ (see Table 1), an arithmetic unit specific for the ALU was designed (see Figure 4).

It receives 8-bit A and B inputs as well as 2-bit FunSel signal and features a 4:1 Multiplexer (MUX) for output selection, a full adder for arithmetic operations and an overflow control sub-circuit.

The design choices and operations can be explained casually as follows:

- 2-bit FunSel Signal

  FunSel signal is being received in two least significant bits of the general FunSel signal, hence they are not to be confused. Because arithmetic operations range in between $(0100 - 0111)_2$ values of the 4-bit FunSel signal, only the least significant two bits were enough for the design. Since the two least significant bits are the ones that change in the given range.

- 4:1 Multiplexer

  A 4:1 MUX was chosen to be added in order to simplify the unit. Even though the unit can be designed using only the 8-bit output of the full adder, the team decided to implement a 4:1 MUX so as to not over-complicate the circuit.

- Arithmetic Operations

  For the FunSel signal "00" no additional operations were needed. Because the output is equal to the 8-bit input A, it is connected straight to the first input of the MUX.

  An addition operation was needed for "01" and "10" signals. The only difference was that the first operation didn't include the carry bit in the summation while the second operation did. To solve this issue, the team used a simple but effective approach. When the most significant bit (MSB) of the 2-bit FunSel signal is "1" the carry in signal is passed as the carry of the summation by using an AND gate and an OR gate.

Lastly, the "11" signal required a subtraction operation. In order to subtract two 8-bit inputs, first 1's complement of input B is taken. This is achieved via the FunSel signal, an 8-bit sign extender and an 8-bit XOR gate. Carry in is then passed in as "1" in order to get the 2's complement of input B. Afterwards the operation is carried via the full adder to get the desired the result.

- Overflow Control Sub-circuit

This part of the unit checks four necessary bits from the circuit in order to decide whether to raise the overflow flag or not. These inputs are, most significant bits of input A and B, the operation bit ($Subtraction/\overline{Addition}$) and the most significant bit of the output.

Because there are four overflow conditions (see Table 2), the sub-circuit checks for the relation among aforementioned four necessary bits. To advert the mechanism, first, XOR gate tells if the MSBs of the inputs are same or not. The NOR gate following the XOR gate, then checks the operation. If the MSBs of the inputs are same and the operation is addition, or if the MSBs are different and the operation is subtraction, then further information from the relation between MSBs of the input B and the output is required.

Second XOR gate then gives the mentioned information. If the conditions above are satisfied and when the operation is addition, the MSBs of the input B and the output have different signs or when the operation is subtraction, the MSBs of the input B and the output have the same signs, the overflow flag is raised. These if-or statements are implemented using a 2 input AND gate, a 3 input AND gate and a 2 input OR gate.

As a result, the arithmetic unit of the ALU performs the way it is intended to and does so very efficiently, thanks to explained design techniques.

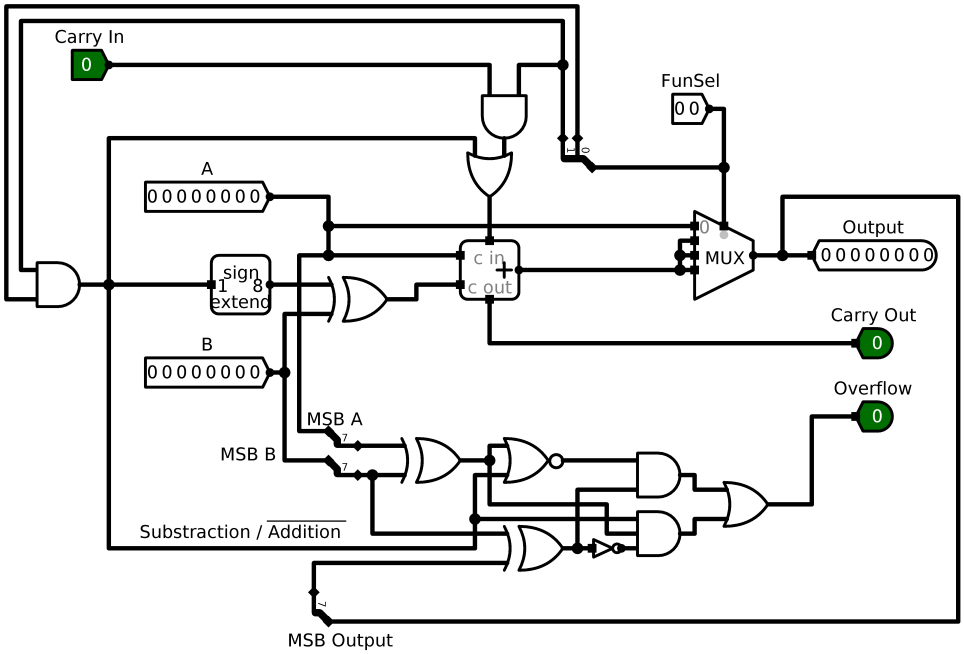

Figure 4: Arithmetic unit of the ALU

Operation	**Result**
pos + pos	neg
neg + neg	pos
pos - neg	neg
neg - pos	pos

Table 2: Overflow conditions

2.1.2 Part-1b

In this sub-part of the arithmetic logic system, shifting functions of ALU are implemented. These functions consist of Arithmetical, Logical and Circular shifting for both directions, right and left. In order to obtain the results given in Table 3, the circuit in Figure 6 is designed.

As it can be be seen in Table 3, bits ranging from 1 - 6 are changes only by the shift direction. Thus they are common for all operations. This is shifting achieved by simply connecting corresponding cables to each other as they are given in Table 3. Values of least and most significant bits are varies over shifting method.

Logical shifting inserts a 0 zero to place that is left empty after shifting. Arithmetic shifting sets LSB to 0 after shifting left, MSB to last MSB after shifting to the right.

Circular shifting sets LSB to last MSB if shifted to left and sets MSB to last LSB if shifted to right. Logical shifting and circular shifting sets the carry bit to last MSB if shifted to left and last LSB if shifted to right.

Some shifting methods can raise overflow as it can be seen in Table 1. This only happens when the sign bit changes. This is checked by XOR in the both most significant bits before and after shifting.

Position	Input	LSL	LSR	ASL	ASR	CSR	CSL
0(LSB)	A_0	0	A_1	0	A_1	A_7	A_1
1	A_1	A_0	A_2	A_0	A_2	A_0	A_2
2	A_2	A_1	A_3	A_1	A_3	A_1	A_3
3	A_3	A_2	A_4	A_2	A_4	A_2	A_4
4	A_4	A_3	A_5	A_3	A_5	A_3	A_5
5	A_5	A_4	A_6	A_4	A_6	A_4	A_6
6	A_6	A_5	A_7	A_5	A_7	A_5	A_7
7(MSB)	A_7	A_6	0	A_6	A_7	A_6	A_0
Carry		A_7	A_0	-	-	A_7	A_0

Table 3: Output of the shifting unit for every shifting operation

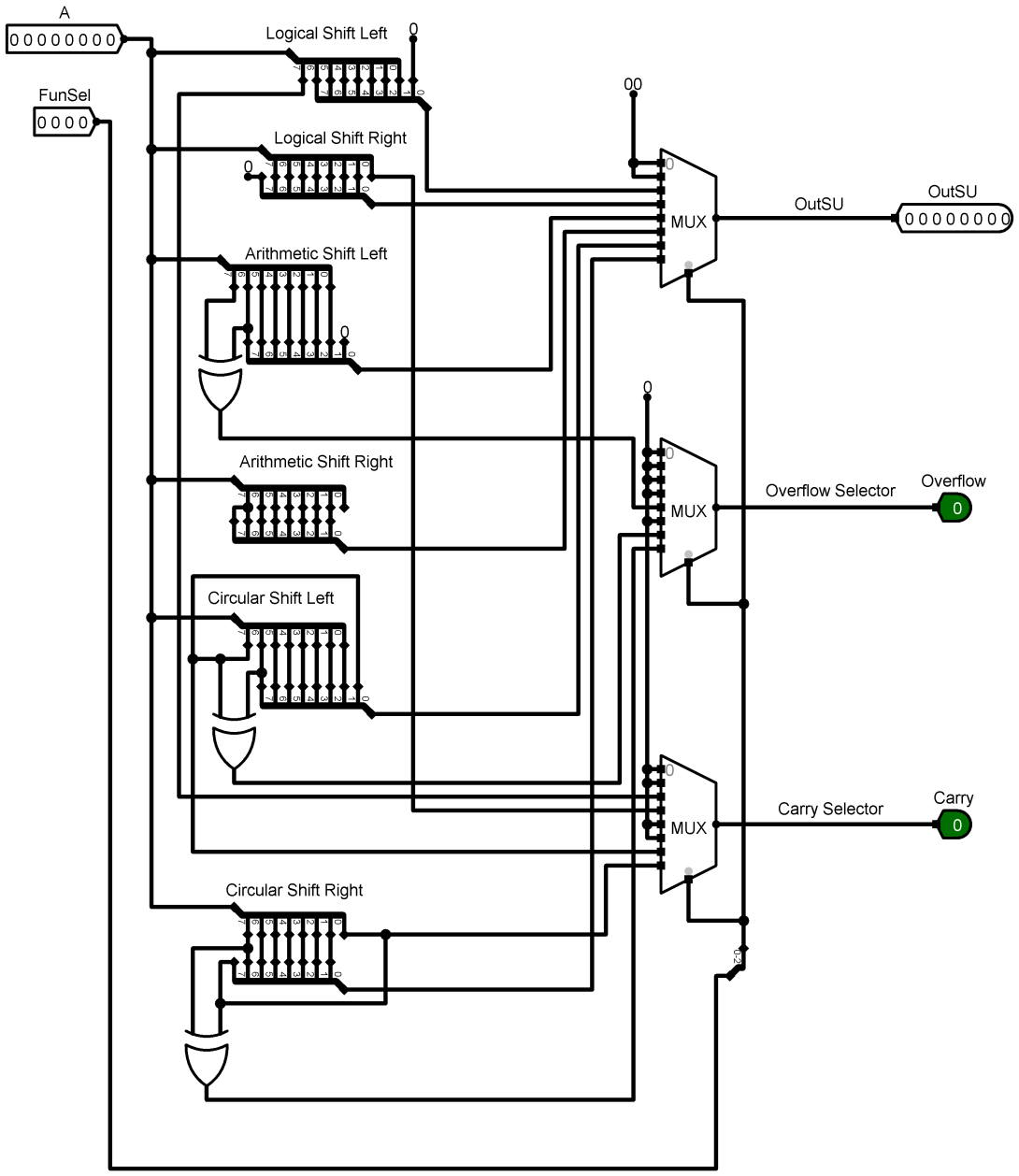

Figure 5: Shift Unit of the ALU

2.1.3 Part-1c

In this section, in order to save the Z(zero), C(carry), N(negative), O(overflow) bits to the memory and send the C(carry) bit to the ALU for the next cycle, a Flag Register is implemented.

In the Flag Register, there is a binary 4-bit input, which represents Z(zero), C(carry), N(negative), O(overflow) and four D flip-flops to save the corresponding bits to the memory.

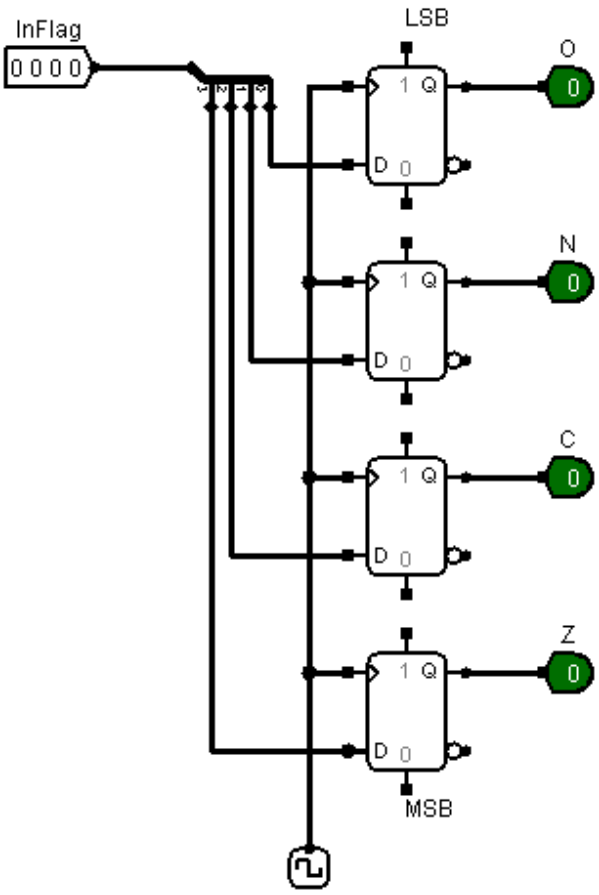

Figure 6: Flag Register of the ALU

2.2 Part-2

In the ALU system, the ALU and the Flag Registers are designed in this project. Two 4:1 MUX gates, a RAM, registers -implemented from files (Register File, Address Register File)- and Instructions Register that are designed in the previous project are used.

MuxASel	**MuxAOut**
00	IROut(0-7)
01	Memory Output
10	Adress
11	OutALU

Table 4: Characteristic table of MuxA inside the ALU System

MuxBSel	**MuxBOut**
00	ϕ
01	IROut(0-7)
10	Memory Output
11	OutALU

Table 5: Characteristic table of MUXB inside the ALU System

The functionality of the components that compose the ALU System can be explained as follows:

- Register File

  With respect to the inputs OutASel and OutBSel, the register file gives the chosen outputs that are created by the registers which located inside the register file. FunSel determines the operation which will be applied to the register (or registers) that are chosen by the RegSel.

- Address Register File

  With respect to the input OutCSel, the address register file gives the chosen output that is currently inside the chosen register which is located inside the address register file. FunSel determines the operation that will be done to the register that is chosen by the RegSel.

- Instructions Register

  With respect to the L/R' input, the IR takes the input to its Lower(0-7) or Higher(8-15) bits and applies the corresponding operation that is chosen by the FunSel

- 4:1 MUX gate A

  With respect to its selector input, it selects one of the following; the lower bits(0-7) of IR, the Memory Output which comes from the RAM, the Address which comes from the Address Register or the previous output which comes from the ALU and sends it to the Register File.

- 4:1 MUX gate B

With respect to its selector input, it selects one of the following; the lower bits(0-7) of IR, the Memory Output which comes from the RAM or the previous output which comes from the ALU and sends it to the Address Register File.

- Flag Register

Gets the OutFlag which comes from the ALU and writes it to its memory and sends the C (carry) bit to the ALU for the next cycle.

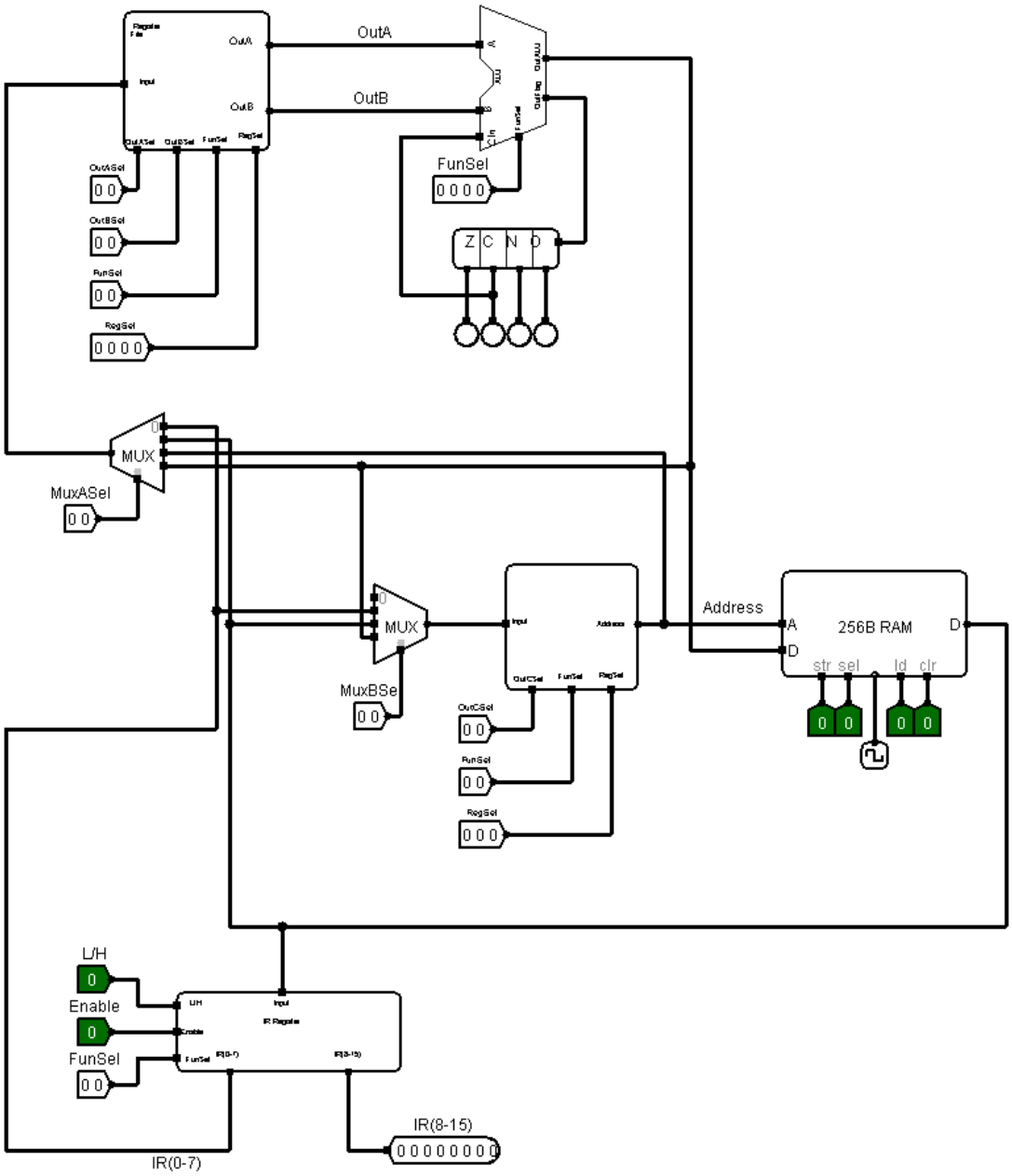

Figure 7: Organization of the ALU System

3 DISCUSSION

During this project, implementation of already designed register and register files in the previous project into more complex structures is learned. Furthermore, designing the ALU with respect to characteristic table was done for the first time by all of the team members. Even though the ALU that is created in this project was simple, working on this project is useful to grasp the simple logic behind an ALU.

During the design process, the project is split into 3 main parts according to the operating units. For instance, these units are logical operating, arithmetic and shifting units. The main reasons for doing so are to be able to do the division of labor effectively and to finish the project quickly, yet efficiently.

After finishing Part-1, all the pieces were put together according to the schematics given in the project description. In the first part, how the ALU and several type of registers can be a part of a simple computer organization was seen.

Generally, team tried to do project in the simplest and the most effective way as much as possible without over-complicating the design. At the end, whatever looks good to the eye is easier to understand.

4 CONCLUSION

In conclusion, it can be said that Project-2 is a following stage for the first project, if the first part of this project is neglected. Because, in the first part, Arithmetic Logic Unit is designed, contradistinctive to the Project-1.

ALU is a unit that used to perform arithmetic and logic operations in digital circuits. It is also known as a fundamental part for the CPU, like a liver in a human. In despite of modern CPU's powerful and complex ALUs, the designed ALU for this project is very basic. It basically has two 8-bit inputs, 4-bit FunSel inputs, and an 8-bit outputs. Afterwards, the ALU is used during the second part of this project in order to implement a simple computer organization. The registers that are created in the previous project are also used in the second part. After implementing a simple computer organization in Part-2, the basic operations becomes possible.

In conclusion, every aspect of the project 2 is designed, tested and confirmed that it functions along the lines of the project description. All the requirements are met and the design functions flawlessly, yet it is quite efficient.

REFERENCES

[1] Overleaf documentation https://tr.overleaf.com/learn.

[2] Detailed info on writing reports https://projects.ncsu.edu/labwrite/res/res-studntintro-labparts.html.

[3] Website lets collabration over projects. https://github.com/.