ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT 2

GROUP MEMBERS:

150180921 : EARTA JOCA

150180922 : INES MUKA

150180924 : GRETA ISARAJ

150180905 : FATIMA RAHIMOVA

SPRING 2020

Contents

1 PART 1

In the first part of this project Arithmetic Logic Unit (ALU) that has two 8-bit inputs and an 8-bit output was designed.

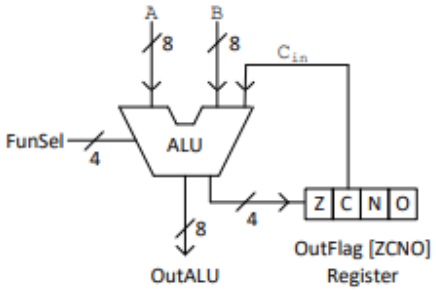

Figure 1: Arithmetic Logic Unit (ALU)

Depending on the FunSel input one of the functions in Figure 2 will be selected.

- **FunSel** selects the function of the ALU.
- **OutALU** shows the result of the operation that is selected by **FunSel** and applied on A and/or B inputs.
- **Z(zero)** bit is set if **OutALU** is zero (e.g., **NOT** B is zero).
- **C (carry)** bit is set if **OutALU** sets the carry (e.g., **LSL A** produces carry).
- **N (negative)** bit is set if the ALU operation generates a negative result (e.g., **A–B** results in a negative number).
- **O (overflow)** bit is set if an overflow occurs (e.g., **A+B** results in an overflow).
- Note that **Z|C|N|O** flags are stored in **a register**!

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 2: Characteristic Table of the Arithmetic Logic Unit (ALU)

Note on figure 2 - the overflow is activated for ASL A and not ASR A

ALU can carry out 16 operations based on the 4 bit FunSel. A 16:1 MUX may be used to reflect the desired result on the output bits and FunSel will be fed to its selector lines. The desired operations will be carried out before and then they will be fed to the MUX, so the only thing it will do is buffer the correct value based on the FunSel combination (the operation that it was required to perform). In this project the numbers are being represented by signed 2's complement, the sign denoted by the MSB (Most Significant Bit). 0 shows a positive number while 1 a negative number.

To make the circuit more organized, we decided to separate the operations into different components that were then loaded into the project as Logisim libraries. There are 3 components that take care of the operations in Figure 2: the Arithmetic unit, the Logic unit and the Shifting unit.

- Logic Unit of the Arithmetic Logic Unit (ALU)

For performing A , B , NOT A , NOT B , A AND B , A OR B , A XOR B functions, this unit of the ALU will be used. We used NOT , AND , OR , XOR gates to implement the circuit. A, B operations are just a direct connection between the input and the output while the NOT operations require a NOT logic gate, so that is why we decided to

incorporate them here. The input of the logic unit are the 8 bits coming from input A and the 8 bits coming from input B. In the output we have the results of each operation carried out for each bit. For the first bit, with index 0, OUTA0 is the A0 bit, OUTB0 is the B0 bit, NOTA0 is the A0 bit fed into a NOT gate, NOTB0 is the B0 bit fed into a NOT gate, AND0 is A0 AND B0 operation carried out using the AND logic gate, OR0 is A0 OR B0 and XOR0 is A0 XOR B0. This was repeated for each bit. Each operation has its specific output.

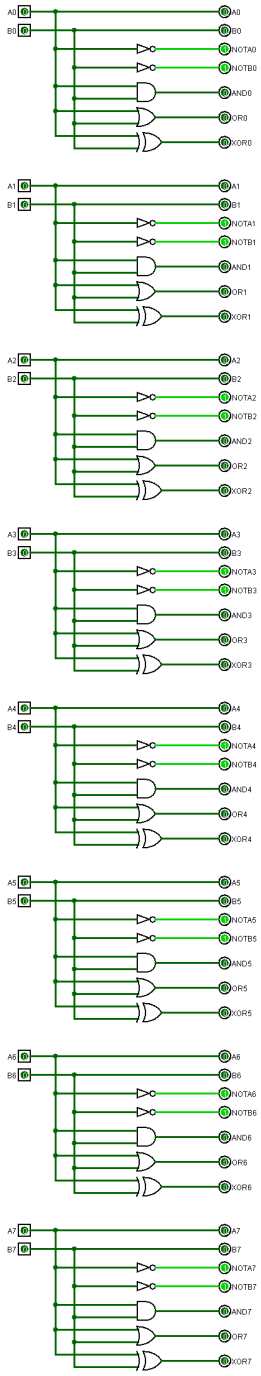

Figure 3: Logic Unit of the Arithmetic Logic Unit (ALU)

- **Arithmetic Unit of the ALU**

For the arithmetic unit, the 3 operations could be performed by a single parallel adder subtractor, with no need of repeating the same circuit three times for each operation. In contrast to the Logic Unit which had an output for each of the operations on each bit, a total of 56 bits in the output, the arithmetic unit only has 8 bits as output of operations (and 1 for the carry out), as it chooses which one it has to perform. The unit has 8 full adders, and in each full adder A is fed and B XOR s1. If we see from the table, among the 3 operations, if s1 = 0, addition is being performed (carry or no carry) and if s1 = 1, subtraction is being performed. For subtraction: A - B = A + (-B). So we add A to B's 2's complement form. If addition is being performed, B XOR s1 will be B XOR 0 which will act as a buffer of B, while if s1 is 1, B XOR s1 becomes B XOR 1, which gives us the 1's complement of B. To complete the conversion to 2's complement we still need to add 1 as carry in. To deal with the value that has to be fed as carry in for these operations, (the carry in for the first of the full adders) we have used two 2:1 MUXes. If we see the table again, if s0 = 0 we don't consider the carry value that comes from the carry flag, while if s1 = 1, we take it into consideration and add it to our number. For one of the MUXes we feed s0 as the select line. The I0 input depends on the operation being performed. It is 0 if its addition and it should be 1 if it's subtraction to complete the transition to 2's complement form. So we put another MUX here which uses s1 as a select line and forwards the right value to the second MUX. The I1 of the s0 select line MUX is the carry value of the carry flag. With these operations we have ensured that the correct value will be added as carry in and the 3 arithmetic operations have been performed successfully. As mentioned above, the carry out value of the operation chosen is 1 of the outputs of this unit.

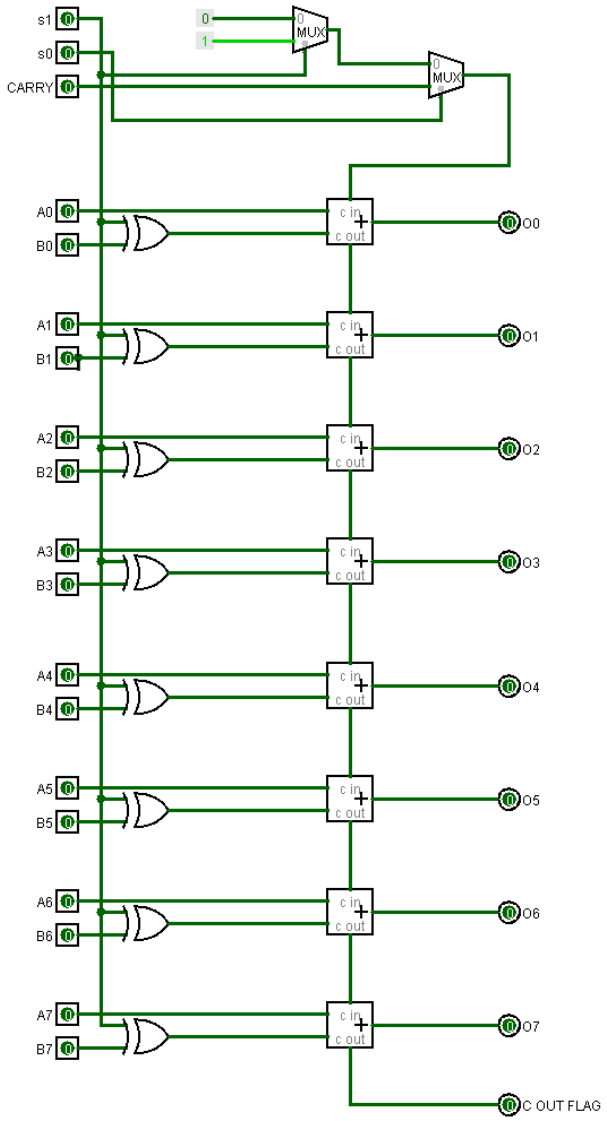

Figure 4: Arithmetic Unit of the Arithmetic Logic Unit (ALU)

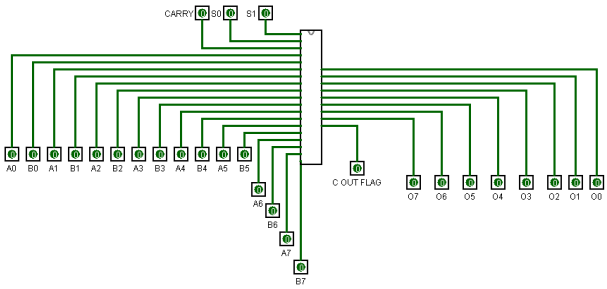

Figure 5: Pin configuration of Arithmetic Unit

- Shifting Unit of Arithmetic Logic Unit (ALU)

In order to implement the shifting functions which are represented by the last 6 functions in Figure 2 we used Shifting Unit of ALU. We can divide classify these functions in the following way:

- Logical Shift Operations;
- Arithmetic Shift Operations.
- Circular Shift Operations;

Here we can repeat the 3 types of shifts in both directions using only one circuit and not repeating it, by choosing the direction in which the shifting will happen (left or right) and what the type of shift through some previous manipulation. To perform the shift operation there are 8 MUXes. With the correct connections, choosing one input of the MUX performs shift left and choosing the other performs shift right. We see from the table that when s0 is 0, shift left is being performed and when s0 is 1, it's shift right, so s0 will act as the select line of the MUXes which give O0 to O7 bits (the result of the shifting). Now we deal with the serial input. For shift left: Logical Shift - 1010 combination Arithmetic Shift - 1100 Circular Shift - 1110 For Logical shift and Arithmetic Shift we feed a 0 as input, so 0 takes the place of the LSB (Least Significant Bit). For Circular Shift we feed the carry value from the carry flag as input, so the carry value takes the place of the LSB. We can use a MUX which has 0 as its first input and carry in as the second, so it sends either one or the other as the serial input. Since s3 is 1 for all the shift operations, we look at the s2s1s0 bits. For the select line of the MUX, the combinations 010 and 100 should give a 0 and the combination 110 should give a 1. This can be achieved by ANDing s2 with s1 and then ORing the result with s0. : (s2 AND s1) OR s0. With 010 and 100 we get a 0, while with 110 we get a 1.

For shift right: Logical Shift - 1011 Arithmetic Shift - 1101 Circular Shift- 1111 For the Logical Shift, we feed 0 as the serial input again, and 0 takes the place of the MSB. For the Arithmetic Shift, we feed the value of A7 (the most significant bit), meaning that the position of A7 preserves its current value and not only shifts it to A6. For the circular shift we feed the value of the carry flag. So here we don't have only 2 options as above, but 3. We first use a MUX to choose between the arithmetic and circular shift. These two differ by their s1 bit, so we feed s1 as the select line. For s1 = 0, it's the arithmetic shift so we connect A7 to I0, while for s1 = 1, it's the circular shift, so we connect the carry value that comes as input to this unit. Afterwards, we use another MUX to choose

between the Logical Shift and the other 2. They differ by s2, so we use s2 as the select line of this second MUX. s2 = 0 means we are performing Logical Shift so we feed 0 as the I0, and s2 = 1 means we are performing one of the other two shifts, so we feed the result of the previous MUX (where we made the choice) to the I1.

It seems a bit complex but this is done to reduce circuitry and not repeat components unnecessarily.

In this shifting unit we also have to decide on the value of the carry out. No carry out is involved in the arithmetic shift. In both the logical and the circular shift the carry out has the same value depending on the direction. If it's left, the carry out has the value of A7 and if it's right, it has the value of A0. So the one who decides the carry out is s0 (since it decides direction). We feed s0 as the select line of a MUX. s0 = 0 means left so we connect A7 to I0 and s0 = 1 means right so we connect A0 to I1. We note here that there will be a carry out value even for arithmetic shift with this setup. But since the carry out flag will be disabled for the combination that selects the arithmetic shift, whatever value we feed to it, it won't make any difference as it will continue to preserve the value it had and not change its state.

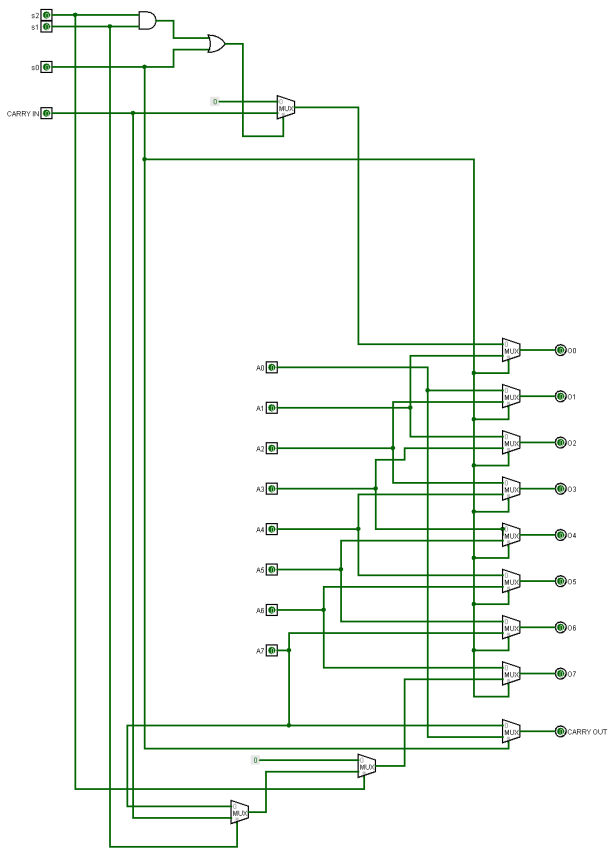

Figure 6: Shifting Unit of Arithmetic Logic Unit(ALU)

- **OutFlag [Z|C|N|O] Register of Arithmetic Logic Unit (ALU)**

We also create the ZCNO register, where we decide on the values of the flags. We have 4 flags, and we have to remember their values, so we use 4 D flip flops to store them. For the zero flag: It will always be activated so the enable input of the flip flop will be always 1. For a number to be 0 it means that all the bits are equal to 0. We use a NOR gate where we connect all the OutALU bits(from OutALU0 to OutALU7). If all the bits are 0, the result will be 1 so the value of the zero flag will be equal to 1 (the OutALU is zero) and for all other combination of bits the NOR gate will give a value of 0 which will be stored in the zero flag (meaning the number is not 0).

For the carry flag: There are cases when the flag won't be enabled. We use a MUX to get the enable input of the flip flop that holds the carry. Referring to the Figure 2, for the combinations where the carry flag should be disabled we connect a 0 to the input of the MUX and for the cases where it should be enabled we connect a 1 to the corresponding input positions of the MUX. The select lines are FunSel. Then we decide the value that the carry flag will store. We use another MUX for this which will forward the right value according to the FunSel combinations. For the cases where the flip flop will be disabled, we can just feed a 0 as it won't make any difference since the state of the flip flop won't change. For the other cases: To the 3 input positions of the MUX that correspond to the combinations regarding the arithmetic operations, we feed the same carry that comes as the carry out of the arithmetic unit we constructed. This means that for a specific FunSel combination, the 3 positions will have the same value, since we are feeding the same thing, even though only 1 of the operations is being carried out, and not the 3 of them. But it's okay since only one value will be reflected at the output of the MUX and this will be the one corresponding to the operation being carried out (depending on the FunSel combination). We see only the correct value at the output. For the 4 input positions of the MUX that correspond to Logical shift and Circular shift (both left and right) we feed the carry out value coming from the shifting unit. The same thing applies here, the same value will be fed in all 4 positions, but it will be the one corresponding to the operation that we are carrying out, so it will be the correct one.

For the negative flag: The flip flop carrying the negative flag should only be disabled for the ASR A - 1101 combination. That's why in the MUX that controls the enable of the Negative flip flop we set all inputs to 1 and only this position to 0. The value of the flip flop is equal to the value of the MSB(most significant bit). If it is 0, the number is positive and if it is 1, the number is negative and the flag is set to 1.

For overflow flag: As for the previous cases, we have a MUX that controls the enable of the flip flop. 0 is inputted for the cases it is disabled and 1 for the cases it is enabled.

Regarding the values of the overflow: When we perform addition in the arithmetic unit, we have overflow in these cases:
(+) + (+) = (-)
(-) + (-) = (+)
The sign of A is shown by A7 and the sign of B by B7. As we see above, we can only have overflow when A and B have the same sign, meaning A7 and B7 carry the same value, so we control their equality using an XNOR gate. It gives a 1 only if they are equal, and 0 otherwise. Also the sign of the OutALU is different from that of the inputs. So we check them being different by an XOR gate where we feed OutALU7 and A7. The gate gives 1 if they have opposite signs, and 0 otherwise. Both conditions must be true for overflow to occur so we AND them together and connect it as input to the corresponding MUX position. This is the same both for addition with and without carry.

When we perform subtraction in the arithmetic unit, we have overflow in these cases:
(-) - (+) = (+)
(+) - (-) = (-)
We can only have overflow when A and B have different signs so we compare A7 and B7 with an XOR gate which gives 1 when they are different and 0 if they are the same. The output should have the same sign as B. We compare OutALU7 with B7 by feeding them into an XNOR gate which gives 1 if they are equal and 0 if they are different. Again, we need both conditions, so we AND the 2 together and connect to the correct position in the MUX.

Regarding the overflow of the shift operations: Only the ASL, CSL and CSR can have overflow. And it happens when the sign of the result is different from the sign of A. For ASL and CSL, the shift left operations, we perform the check by seeing before the shift whether A7 and A6 have different signs or not. To do this, we feed them to an XOR gate. If they have different signs, this means that the sign will change after the shift and we have overflow. XOR gate will give a 1 which will be connected to the MUX. For the CSR operation, since the carry stored in the carry flag will become the MSB, we compare that to the sign of A before shifting, represented by A7. We XOR them together and the result will be 1 when they are different, meaning it overflows. Here, we have also included some LED lights just to check the values that the flags take. Also, since flip flops are involved and it is a sequential circuit, we have included the clock signal.

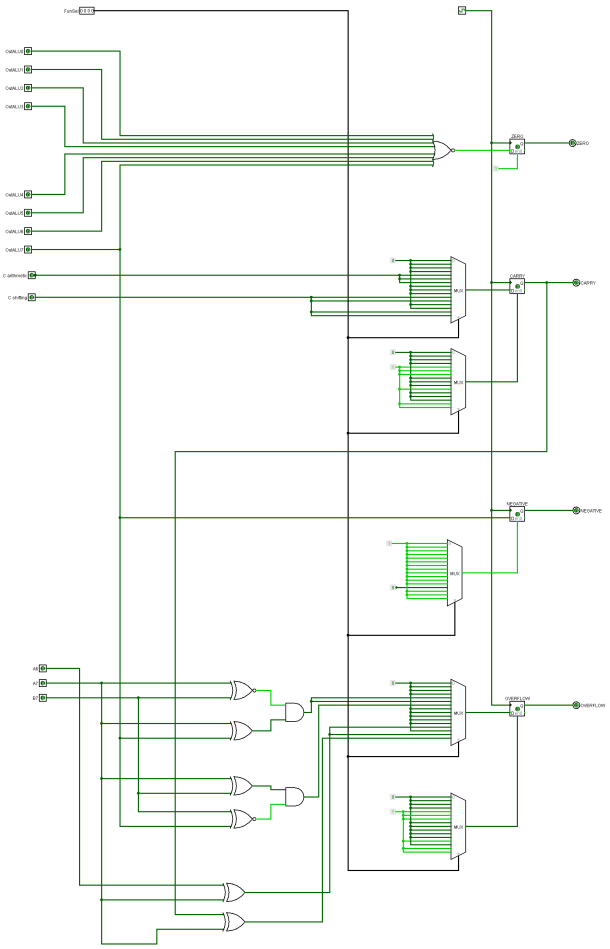

Figure 7: OutFlag Register of Arithmetic Logic Unit(ALU)

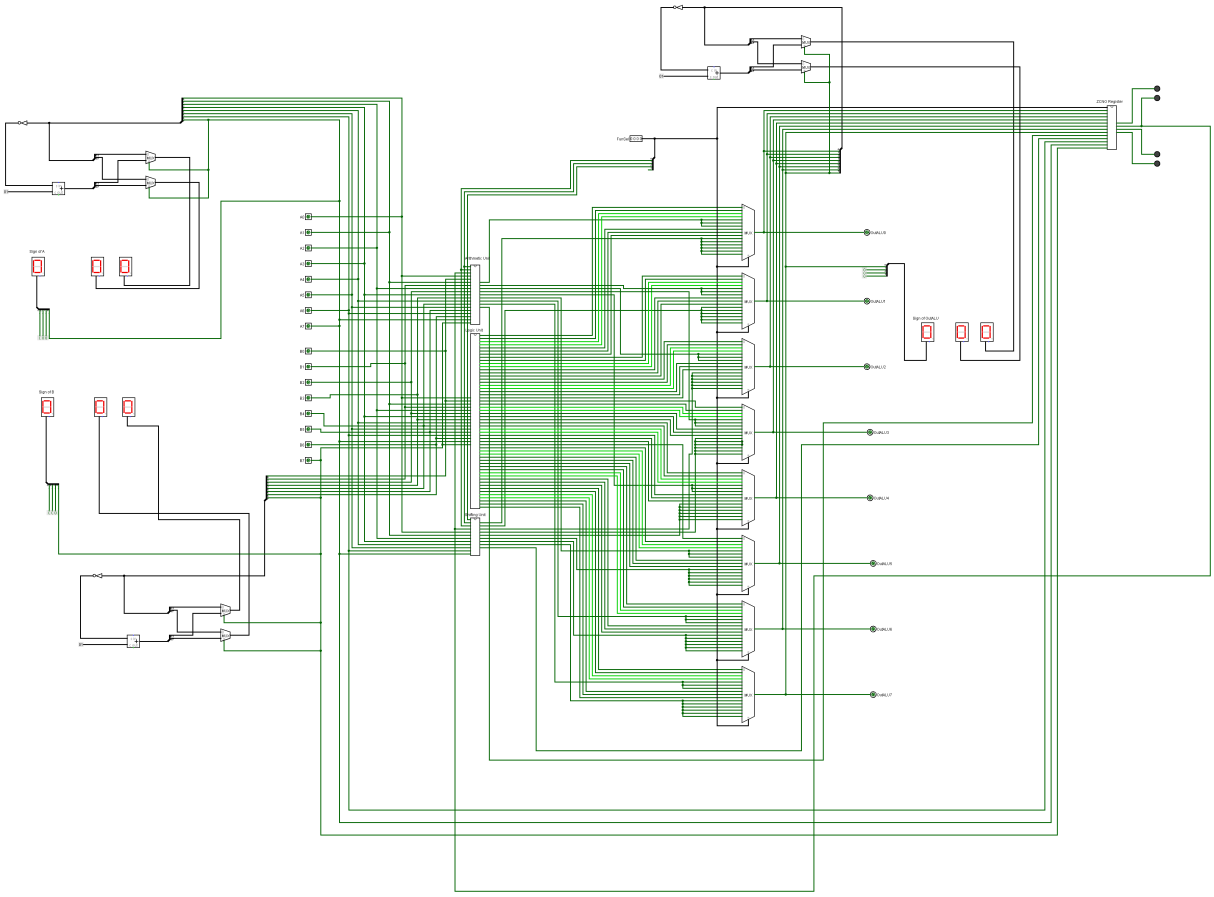

Figure 8: Arithmetic Logic Unit(ALU)

We join all the components in the main circuit of Part 1. The outputs of the units we created are fed to the MUXes which correspond to each OutALU. Hex Digits Displays are connected to the input A and B and also to the OutALU. Since we are dealing with signed 2's complemented representation, if 1 of the numbers is negative, then we need to take its 2's complement to get the displays to show its absolute value. That's why we use a MUX which gets the direct bits in I0 and 2's complement form of the number in I1. For the 2's complement, all the bits are complemented and 1 is added afterwards. The select line is the sign bit of the numbers, their MSB. Since A, B and OutALU are 8 bits, we need 2 HEX Displays, one of which represents the 4 lower bits (the one in the right), and the other in the left the higher bits. An additional HEX Display has been added to show the sign of the number. It needs a 4 bit data input to work, so we add three 0's in front of the MSB, so we don't change its value and then feed it. It will show a 0 if the sign is positive and a 1 if it is negative.

2 PART 2

The register files which designed in the first project , two 4:1 MUX gates , 2:1 MUX gate , separate load RAM , the ALU designed in part 1 of this project are used for implementing the whole ALU system.

MuxASel	MuxAOut
00	IROut (0-7)
01	Memory Output
10	Address Register OutC
11	OutALU

Figure 9: Characteristic table of MuxA

If MuxASel equals to 00 then least significant bits of IR ; in the case of MuxASel equals to 01 the memory output ; when MuxASel equals to 10 OutC of Address Register File and if MuxASel equals to 11 output of ALU will be selected and selected output will be sent to the Register File and input of the MuxC .

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

Figure 10: Characteristic table of MuxB

If MuxBSel equals to 01 then least significant bits of IR ; in the case of MuxBSel equals to 10 the memory output ; when MuxBSel equals to 11 output of ALU will be selected and selected output will be sent to the Address Register File .

MuxCSel	MuxCOut
0	MuxAOut
1	OutA

Figure 11: Characteristic table of MuxC

If MuxCSel equals to 0 then MuxC selects output of MuxA ; in the case of MuxCSel equals to 1 then OutA of the Register File will be selected and the selected output will be sent to the ALU.

- Memory (RAM) is used to storing the address and value information.

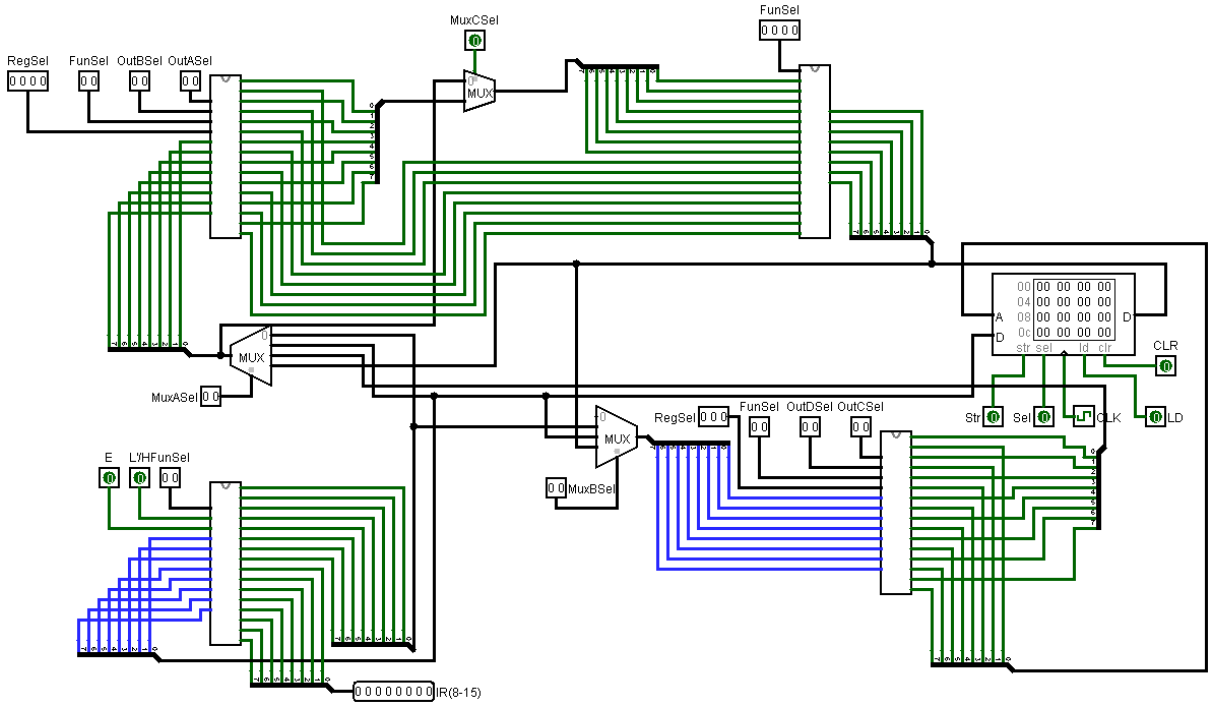

Figure 12: ALU System

REFERENCES

[1] https://ninova.itu.edu.tr/Sinif/3557.42763/Odev/57200?g1768547