ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 1

PROJECT DATE : 10.05.2021

GROUP NO : 24

GROUP MEMBERS:

150180056 : BEYZA OZAN

150180090 : MİHRİBAN NUR KOÇAK

150180099 : MEHMET YİĞİT BALIK

SPRING 2021

Contents

1 INTRODUCTION

In this project, we designed a small processor system. The system is designed as parts. In the first part, 8-bit register and 16-bit instruction register are designed. In the second part, 8-bit general register file and 8-bit address register file are designed. In the third part, an Arithmetic Logic Unit System is designed. In the last part, all the previous parts are combined with a memory.

2 PROJECT PARTS

Main circuit files of the parts and their component circuits:

- `8-bit_Register(Part1a).circ`
  - Component: `8-bit_Adder_Subtractor.circ`
  - Component: `8-bit_D-FlipFlop.circ`
- `16-bit_InstructionRegister(Part1b).circ`
  - Component: `16-bit_Adder_Subtractor.circ`
  - Component: `16-bit_D-FlipFlop.circ`
- `8-bit_GeneralRegisterFile(Part2a).circ`
  - Component: `8-bit_Register(Part1a).circ`
- `8-bit_AdressRegisterFile(Part2b).circ`
  - Component: `8-bit_Register(Part1a).circ`
- `ALUSystem(Part3).circ`
- `WholeSystem(Part4).circ`
  - Component: `8-bit_GeneralRegisterFile(Part2a).circ`
  - Component: `8-bit_AdressRegisterFile(Part2b).circ`
  - Component: `16-bit_InstructionRegister(Part1b).circ`
  - Component: `ALUSystem(Part3).circ`

2.1 Part 1

In this part, we implemented 8-bit Register and 16-bit Register which are fundamental units for our design.

2.1.1 a)

To implement a 8-bit Register and then to obtain its 8-bit output **Q+**, we need a Multiplexer, an Adder-Subtractor, a D Flip-Flop and desired inputs.

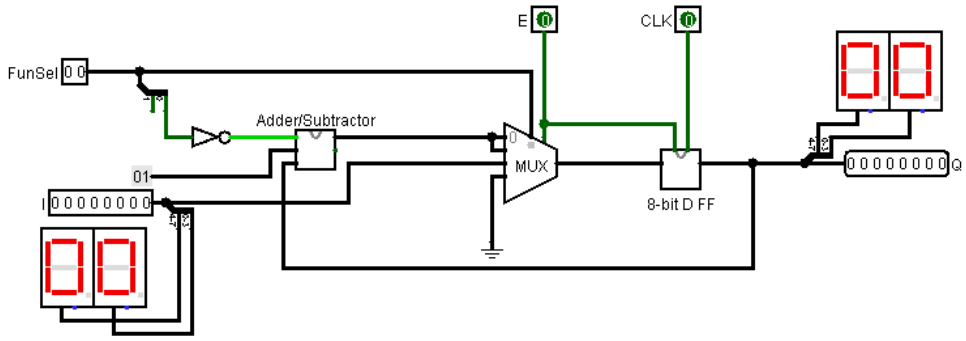

Figure 1: Logisim Circuit For Part-1a

- **Inputs**:
  - **I**: 8-bit input data which becomes the output of the register while doing load operation.
  - **CLK**: the clock signal which synchronize the register.
  - **E**: 1-bit enable input which decides whether the register's functionality is enabled or not. When E is 0, the output of the register(Q+) is remained as current output (Q). Otherwise (E = 1), the register can perform its functionality according to FunSel.
  - **FunSel**: 2-bit control signal which decides the register's functionality while E is 1. The output of the register becomes one minus of its own value while FunSel is 00 which corresponds to Decrement operation, one plus of its own value while FunSel is 01 which corresponds to Increment operation, the I value while FunSel is 10 which corresponds to Load operation and constant 0 value while FunSel is 11 which corresponds to Clear operation.
- **Output**
  - **Q+**: 8-bit output data.

To do selection of functionality according to FunSel input, we have used 4:1 Multiplexer. The inputs of the multiplexer is connected as following:

  * 0 (00): Q-1 which is obtained by sending Q, 01 as operands and FunSel's least significant bit's complement (1) as Adder-Subtracter input to Adder-Subtracter.

* 1 (01): Q+1 which is obtained by sending Q, 01 as operands and FunSel's least significant bit's complement (0) as Adder-Subtracter input to Adder-Subtracter.
* 2 (10): directly to the I input
* 3 (11): directly to the 0 constant

To perform increment and decrement operations we need 8-bit Adder-Subtracter. Here is Adder-Subtracter's behaviour:

* While Adder-Subtracter input is 0, it performs addition operation by sending A and B to 8-bit adder with cin = Adder-Subtracter = 0.
* While Adder-Subtracter input is 1, it performs subtraction operation. First, 1's complement of B input is obtained using 8-bit XOR gate and the A and B is sent to 8-bit adder with cin = Adder-Subtracter = 1.

Finally, based on the knowledge that registers consist of flip-flops, we have used a 8-bit D Flip-Flop to synchronize the register by using clock signal. In this way, the register operate only at the rising edge of the CLK signal.

2.1.2 b)

To implement a 16-bit Instruction Register and then to obtain its 16-bit output **IR+**, we need three Multiplexers, an Adder-Subtractor, a D Flip-Flop and desired inputs:

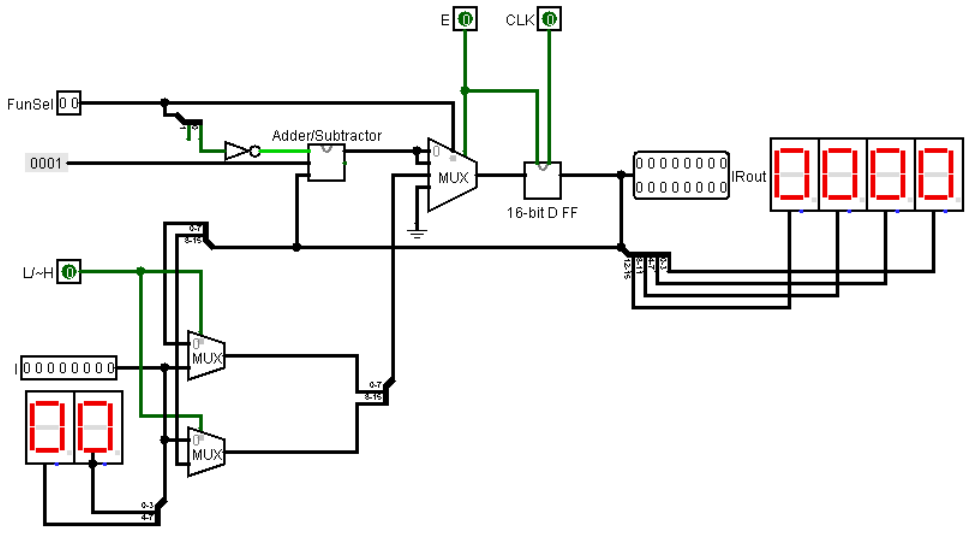

Figure 2: Logisim Circuit For Part-1b

- **Inputs**:

- **I**: 8-bit input which becomes the output's selected portion according to L/H' signal while doing load operation.
- **CLK**: the clock signal which synchronize the register.
- **Enable**: 1-bit enable input which decides whether the register's functionality is enabled or not. When Enable is 0, the output of the register(IR+) is remained as current output (IR). Otherwise (Enable = 1), the register can perform its functionality according to FunSel and L/H'.
- **L/H'**: 1-bit signal which decides to that the I will be loaded to which portion of the IR when FunSel is 10. While FunSel is 10, I is loaded to IR's most significant half (8-15) if L/H' is 0, I is loaded to IR's least significant half (0-7) if L/H' is 1.
- **FunSel**: 2-bit control signal which decides the register's functionality while Enable is 1. The output of the register becomes one minus of its own value while FunSel is 00 which corresponds to Decrement operation, one plus of its own value while FunSel is 01 which corresponds to Increment operation, the I value on its most significant portion (8-15) while FunSel is 10 and L/H' is 0 which corresponds to Load MSB operation, the I value on its least significant portion (0-7) while FunSel is 10 and L/H' is 1 which corresponds to Load LSB operation and constant 0 value while FunSel is 11 which corresponds to Clear operation.

• **Output**

- **IR+**: 16-bit output data.

To do selection of functionality according to FunSel input, we have used 4:1 Multiplexer. The inputs of the multiplexer is connected as following:

* **0 (00)**: IR-1 which is obtained by sending IR, 0001 as operands and FunSel's least significant bit's complement (1) as Adder-Subtracter input to Adder-Subtracter.
* **1 (01)**: IR+1 which is obtained by sending IR, 0001 as operands and FunSel's least significant bit's complement (0) as Adder-Subtracter input to Adder-Subtracter.
* **2 (10)**: directly to the 16-bit data which is the composition of IR and I according to L/H'. If L/H' is 0, the data's most significant portion is I, least significant portion is IR(0-7). If L/H' is 1, the data's most significant portion is IR(8-15), least significant portion is I.

* **3 (11)**: directly to the 0 constant

To compose IR and I according to L/H' signal, we have used two 2:1 Multiplexer. The inputs of the multiplexers is connected as following:

* 0 for above one (0): IR(0-7)
* 1 for above one (1): I input
* 0 for below one (0): I input
* 1 for below one (1): IR(8-15)

To perform increment and decrement operations we need 16-bit Adder-Subtracter. Here is Adder-Subtracter's behaviour:

* While Adder-Subtracter input is 0, it performs addition operation by sending A and B to 16-bit adder with cin = Adder-Subtracter = 0.
* While Adder-Subtracter input is 1, it performs subtraction operation. First, 1's complement of B input is obtained using 16-bit XOR gate and the A and B is sent to 16-bit adder with cin = Adder-Subtracter = 1.

Finally, based on the knowledge that registers consist of flip-flops, we have used a 16-bit D Flip- Flop to synchronize the register by using clock signal. In this way, the register operate only at the rising edge of the CLK signal.

2.2 Part 2

2.2.1 a)

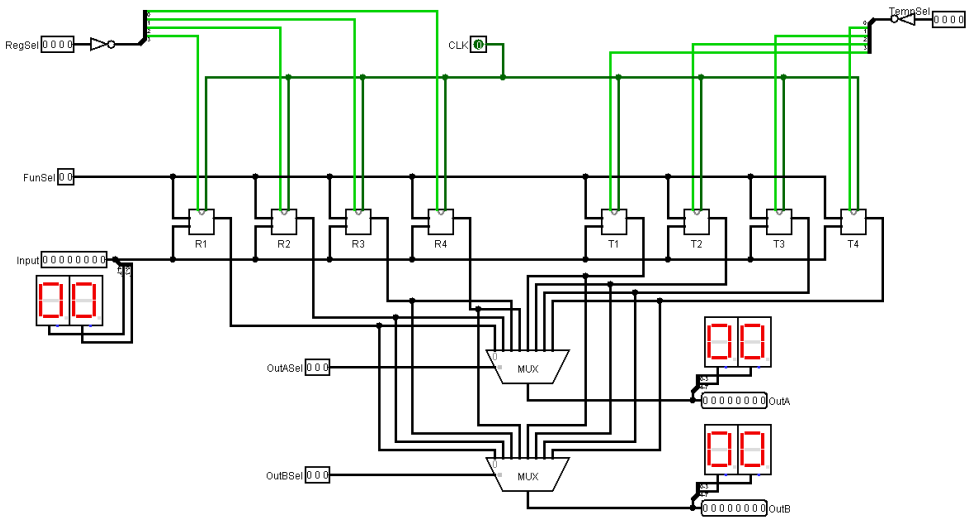

Figure 3: Logisim Circuit For Part-2a

- **Inputs**:
  - Input: 8-bit input data.
  - RegSel: 4-bit register selector to make general purpose register(s) enabled.
  - TempSel: 4-bit register selector to make temporary register(s) enabled.
  - OutASel: 3-bit multiplexer selector input for writing into Output A.
  - OutBSel: 3-bit multiplexer selector input for writing into Output B.
  - FunSel: 2-bit function selector for 8-bit registers.
  - CLK: Clock signal for synchronization.
- **Outputs**
  - OutA: 8-bit output data (Output A).
  - OutB: 8-bit output data (Output B).

In this part, we designed a general register file which consists of 4 8-bit general purpose registers and 4 8-bit temporary registers. For the registers, the 8-bit register which is designed in **Part 1** is used. In our design, we used 2 different 3:8 multiplexers, one for writing to the **OutA** (Output A) and the other one for writing to the **OutB** (Output B). Selector inputs of the multiplexers are **OutASel** and **OutBSel**, respectively. To enable or disable registers, 2 4-bit inputs are used. Those are **RegSel**, which enables or disables general purpose register(s), and **TempSel**, which enables or disables temporary register(s). If the corresponding bit of the **RegSel** or **TempSel** is "0", then corresponding register becomes enabled. For instance, if **RegSel = 0111** and **TempSel = 1110**, then R1 and T4 registers become enabled. All the register are connected to one common clock signal for synchronization.
Both multiplexers are working exactly the same except they write output to different wires. Selector input combinations for multiplexers:

* 0 (000): R1 is written to the output.
* 1 (001): R2 is written to the output.
* 2 (010): R3 is written to the output.
* 3 (011): R4 is written to the output.
* 4 (100): T1 is written to the output.
* 5 (101): T2 is written to the output.
* 6 (110): T3 is written to the output.
* 7 (111): T4 is written to the output.

2.2.2 b)

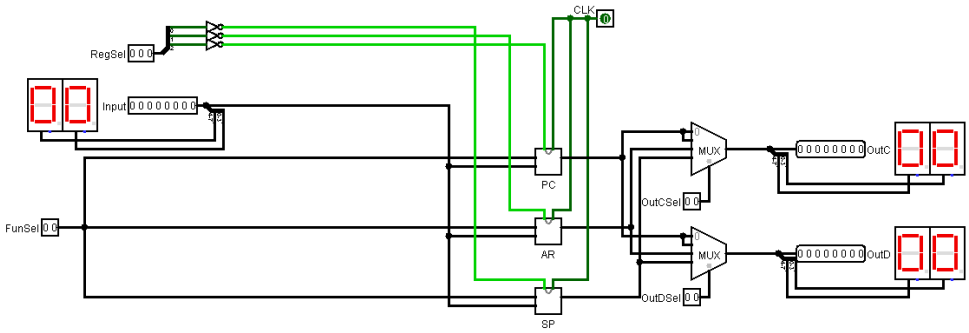

Figure 4: Logisim Circuit For Part-2b

- **Inputs**:
  - Input: 8-bit input data.
  - RegSel: 4-bit register selector to make address register(s) enabled.
  - OutCSel: 2-bit multiplexer selector input for writing into Output C.
  - OutDSel: 2-bit multiplexer selector input for writing into Output D.
  - FunSel: 2-bit function selector for 8-bit registers.
  - CLK: Clock signal for synchronization.
- **Outputs**
  - OutC: 8-bit output data (Output C).
  - OutD: 8-bit output data (Output D).

In this part, we designed a address register file which consists of 3 8-bit registers.For the registers, the 8-bit register which is designed in **Part 1** is used. In our design, we used 2 different 2:4 multiplexers, one for writing to the **OutC** (Output C) and the other one for writing to the **OutD** (Output D). Selector inputs of the multiplexers are **OutCSel** and **OutDSel**, respectively. To enable or disable registers, 3-bit input, which is **RegSel**, is used. Most significant bit is used for **PC**, second bit is used for **AR** and third bit is used for **SP**. If the corresponding bit is "0", then corresponding address register(s) become enabled. For instance, if **RegSel = 010**, then **PC** and **SP** becomes enabled. All the register are connected to one common clock input for synchronization. If the multiplexer selector input is 00 or 01, then the value stored in **PC** is given to the output. Multiplexer combinations for **AR** and **SP** are 10 and 11, respectively.

2.3 Part 3

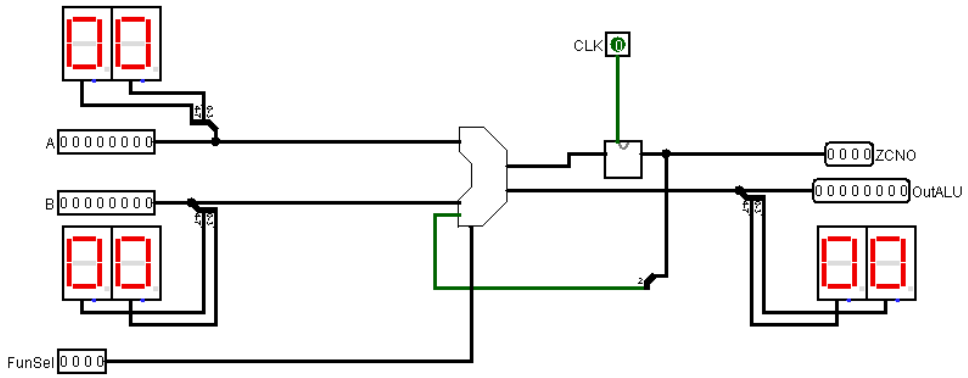

Figure 5: Logisim Circuit For Part-3 (ALU System with ZCNO register)

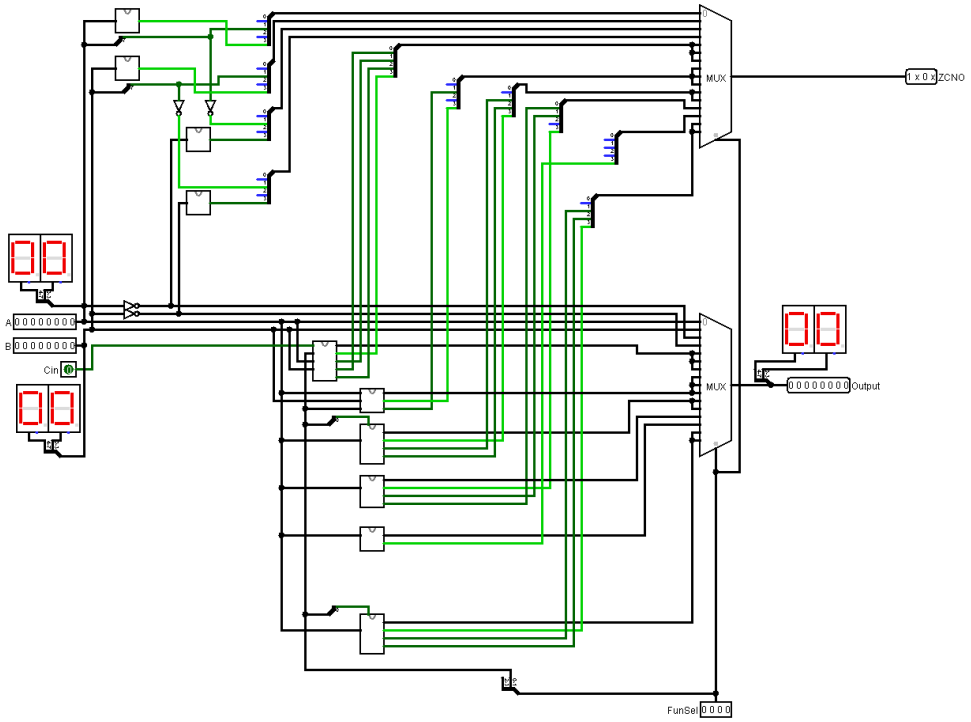

Figure 6: Logisim Circuit For Part-3 (internal structure of ALU)

- **Inputs**:
  - A: 8-bit input data.
  - B: 8-bit input data.
  - FunSel: 4-bit function selector for 16 different functions.
  - CLK: Clock signal for synchronization.
- **Outputs**

– OutALU: 8-bit output data.

– OutFlag (ZCNO): 4-bit output for zero, carry, negative and overflow flags.

In this part, we designed Arithmetic Logic Unit (ALU). Since we have 16 different functions, we used 4-bit input FunSel input to select the function. We used a 4:16 MUX (first) to map the inputs through these 16 functions to OutALU. We used one more 4:16 MUX (second) to map the different ZCNO according to the chosen function to OutFlag. We take carry value from ZCNO output and send to the ALU design as Cin input. We created a zero detector to decide whether the output is zero or not. We also check whether the output is negative or not by looking at the last bit of outputs.

2.3.1 FunSel : 0000, 0001

In this case, the output of the functions is directly the input. Hence, we connect the input A to 0th index of the first MUX and input B to 1st index of the first MUX. In other words, for input A (FunSel = 0000), the output becomes A and for input B (FunSel = 0001), the output becomes B. For OutFlag, we only consider zero (Z) and Negative (N) flags. We check the output whether it is zero and negative or not using zero detector which we designed and the last bit. The other flags are empty. Then we send these 4-bit ZCNO to second MUX. We connect it to 0th index for A and 1st index for B.

2.3.2 FunSel : 0010, 0011

In this case, the output of the functions is the complement of the input. First, we obtain NOT A and NOT B using NOT gate and we connect NOT A to 2nd index of the first MUX and NOT B to 3rd index of the first MUX. In other words, for input A (FunSel = 0010), the output becomes $\sim$ A and for input B (FunSel = 0011), the output becomes $\sim$ B. For OutFlag, we only consider zero (Z) and negative (N) flags. We check the output whether it is zero and negative or not using zero detector which we designed and the last bit. The other flags are empty. Then we send these 4-bit ZCNO to second MUX. We connect it to 2nd index for NOT A and 3rd index for NOT B.

2.3.3 FunSel : 0100, 0101, 0110

In this case, we implemented three arithmetic operations which are : (**A + B**), (**A + B + Carry**) and (**A - B**). For these functions, we designed and used ArithmeticUnit module. In this module, we used eight Adder/Subtractors, nine XOR gates, one 2:4 MUX and one zero detector. Then we connect A, B and least significant two bits of FunSel and Cin to the ArithmeticUnit as input then we connect the output of this ArithmeticUnit to the first MUX as indexes 4, 5 and 6. For OutFlag, we consider all

bits which are zero(Z), carry(C), negative(N) and overflow(O) flags. We send these 4-bit ZCNO output to second MUX. We connect it to 4rd index for (**A + B**), 5th index for (**A + B + Carry**) and 6th index for (**A - B**).

2.3.4 FunSel : 0111, 1000, 1001

In this case, we implemented three logic operations which are : (**A & B**), (**A** $\|$ **B**) and (**A** $\oplus$ **B**). For these operations, we designed and used LogicalUnit module. In this module, we used a AND, OR, XOR gates and one 2:4 MUX to select the operations. We send A, B and least significant two bits of FunSel to LogicalUnit as inputs. Then we connect the output of LogicalUnit to the first MUX as indexes 7, 8, 9. For OutFlag, we only consider zero (Z) and negative (N) flags. The other flags are empty. Then we send these 4-bit ZCNO to second MUX. We connect it to 7th index for (**A & B**) and 8th index for (**A** $\|$ **B**), 9th index for (**A** $\oplus$ **B**).

2.3.5 FunSel : 1010, 1011

In this case, we implemented logical shift right and shift left operations. First, we designed and used LogicalShift module. In this module, we used eight 1:2 MUX for shifting, one zero detector for zero flag and one more 1:2 MUX for carry flag. We send input A and least significant bit of FunSel for right/left selection to the LogicalShift, then we connect the output of the LogicalShift to the first MUX as index 10 (for left shifting) and 11 (for right shifting). For OutFlag, we consider zero(Z), carry(C) and negative (N) flags. The overflow(O) flag is empty. We send these 4-bit ZCNO output to second MUX. We connect it to 10th index for left shifting, 11th index for right shifting.

2.3.6 FunSel : 1100, 1101

In this case, we implemented arithmetic shift right and left operations. First, we designed and used ArithmeticShiftLeft and ArithmeticShiftRight modules.In the ArithmeticShiftLeft module, we used one zero detector for zero flag and one XOR gate for overflow flag. We send input A to the ArithmeticShiftLeft, then we connect the output of the ArithmeticShiftLeft to the first MUX as index 12 (for left shifting). For OutFlag of left shifting, we consider zero(Z), negative (N) and overflow(O) flags. The carry(C) flag is empty. We send this 4-bit ZCNO output to second MUX. We connect it to 12th index for left shifting.

In the ArithmeticShiftRight module, we only used zero detector because there is only zero flag case. We send input A to the ArithmeticShiftRight, then we connect the output of the ArithmeticShiftRight to the first MUX as index 13 (for right shifting).For OutFlag

of right shifting, we only consider zero(Z). The other flags are empty. We send this 4-bit ZCNO output to second MUX. We connect it to 13th index for right shifting.

2.3.7 FunSel : 1110, 1111

In this case, we implemented circular shift right and left operations. First, we designed and used CircularShift module. In this module, we used eight 1:2 MUX for shifting, one zero detector for zero flag, two XOR gates and one 1:2 MUX for overflow flag, and one more 1:2 MUX for carry flag. We send input A and least significant bit of FunSel for right/left selection to the CircularShift, then we connect the output of the CircularShift to the first MUX as index 14 (for left shifting) and 15 (for right shifting). For OutFlag, we consider all bits which are zero(Z), carry(C), negative(N) and overflow(O) flags. We send these 4-bit ZCNO output to second MUX. We connect it to 14th index for left shifting, 15th index for right shifting.

2.4 Part 4

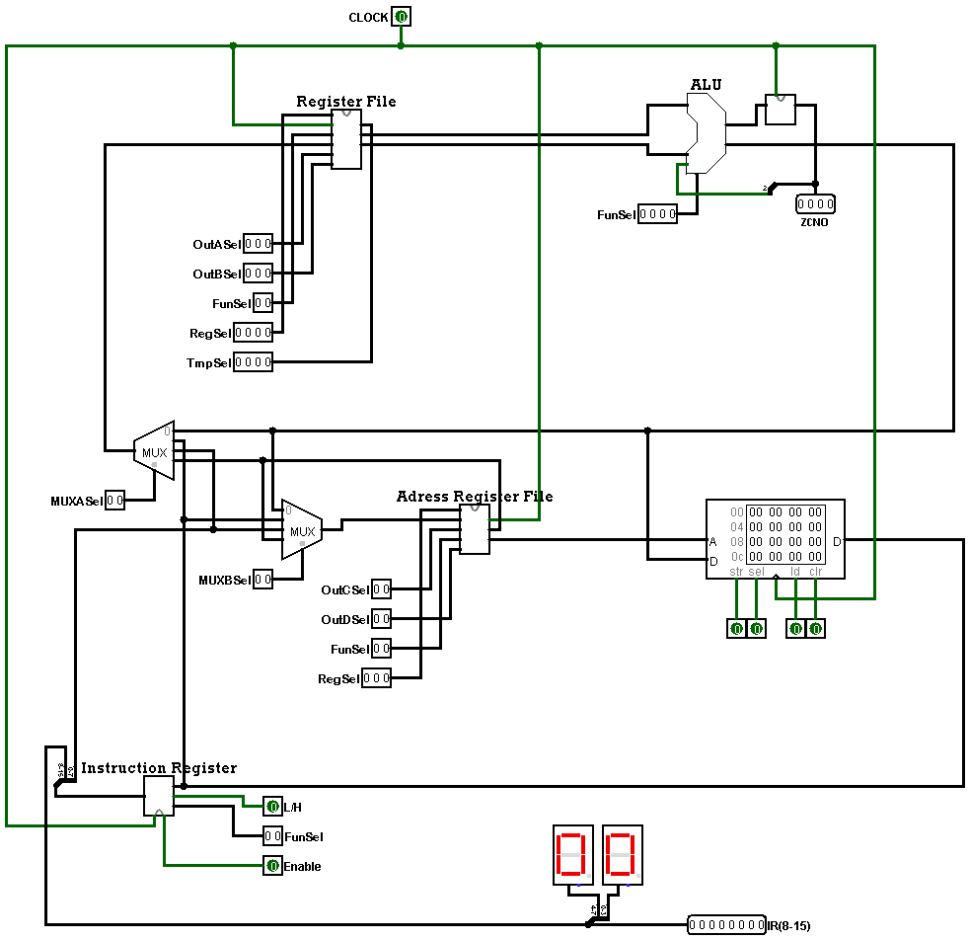

Figure 7: Logisim Circuit For Part-4

In this part, we implemented an organization which consist of all of the units we designed in previous parts. This organization consists of the following units:

* **Address Register File** was designed in Part-2b.
* **Memory(RAM)** was received from pre-prepared design of logisim.
* **Instruction Register** was designed in Part-1b.
* **Register File** was designed in Part-2a.
* **Arithmetic Logic Unit(ALU)** was designed in Part-3.
* **MUX A** was used for selecting input of Register File. output of ALU was connected to 0th index, output of RAM was connected to 1st index, output of Instruction Register's least significant 8-bits was connected to 2nd input, Adress Register File's one of the outputs which is labeled as OutC was connected to 3nd index of 4:1 multiplexer as inputs.
* **MUX B** was used for selecting input of Register File. output of ALU was connected to 0th index, output of RAM was connected to 1st index, output of Instruction Register's least significant 8-bits was connected to 2nd input, Adress Register File's one of the outputs which is labeled as OutC was connected to 3nd index of 4:1 multiplexer as inputs.

- **Inputs**:
  - For Adress Register File:
    * RegSel: 4-bit register selector to make address register(s) enabled.
    * OutCSel: 2-bit multiplexer selector input for writing into Output C.
    * OutDSel: 2-bit multiplexer selector input for writing into Output D.
    * FunSel: 2-bit function selector for 8-bit registers.
    * CLK: Common clock signal for synchronization.
  - For Memory:
    * Store : 1-bit, if 1, store input to memory.
    * Chip Select : 1-bit, 0 disables component.
    * Load : 1-bit, if 1, load memory to output.
    * Clear : 1-bit, when 1, resets contents to 0 asynchronously.
    * CLK: Common clock signal for synchronization.
  - For Instruction Register:

    - Enable: 1-bit enable input which decides whether the register's functionality is enabled or not.
    - L/H': 1-bit signal which decides to that the I will be loaded to which portion of the IR when FunSel is 10.
    - FunSel: 2-bit control signal which decides the register's functionality while Enable is 1.
    - CLK: Common clock signal for synchronization.
  - For Register File:
    - RegSel: 4-bit register selector to make general purpose register(s) enabled.
    - TempSel: 4-bit register selector to make temporary register(s) enabled.
    - OutASel: 3-bit multiplexer selector input for writing into Output A.
    - OutBSel: 3-bit multiplexer selector input for writing into Output B.
    - FunSel: 2-bit function selector for 8-bit registers.
    - CLK: Common clock signal for synchronization.
  - For ALU:
    - FunSel: 4-bit function selector for 16 different functions.
    - CLK: Clock signal for synchronization.
  - For MUX A:
    - MuxASel: 2-bit output selector.
  - For MUX B:
    - MuxBSel: 2-bit output selector.

- **Output**
  - IR(8-15): 8-bit output, most significant 8-bit of Instruction Register's output.
  - ZCNO: 4-bit output, OutFlag output of ALU for zero, carry, negative and overflow flags.

Finally, to complete our design, we made following inter-circuit connections:

- MUXB's output was connected to Adress Register File's Input port as input.
- Adres Register File's OutD output was connected to Memory's Adress(A) port as input.
- ALU's output was connected to Memory's Input(D) port as input.
- Memory's Data output was connected to Instruction Register's I port as input.

* MUXA's output was connected to Register File's Input port as input.
* Register File's OutA output was connected to ALU's A port as input.
* Register File's OutB output was connected to ALU's B port as input.

3 RESULTS

During the experiment we have tested each individual part by different input combinations.

3.1 Part 1

3.1.1 a)

(i) When E = 1, I = 10000000 (0x80) and FunSel = 10, if rising edge of the clock signal arrives, output becomes Q+ = 1000000 (0x80) which means load operation was done.

(ii) When E = 1, FunSel = 00 and Q = 10000000 from previous operation, if rising edge of the clock signal arrives, output becomes Q+ = 01111111 (0x7F) which means decrement operation was done.

(iii) When E = 1, FunSel = 01 and Q = 01111111 from previous operation, if rising edge of the clock signal arrives, output becomes Q+ = 10000000 (0x80) which means increment operation was done.

(iv) When E = 1, FunSel = 11, if rising edge of the clock signal arrives, output becomes Q+ = 00000000 (0x0) which means clear operation was done.

(v) When E = 0 and Q = 00000000 from previous operation, if rising edge of the clock signal arrives, out becomes Q+ = 00000000 which is the value of Q regardless of FunSel.

3.1.2 b)

(i) When E = 1, I = 10000000 (0x80), L/H = 0 and FunSel = 10, if rising edge of the clock signal arrives, output becomes IR+ = 1000000000000000 (0x8000) which means Load MSB operation was done.

(ii) When E = 1, I = 10000000 (0x80), L/H = 0, FunSel = 10 and IR = 100000000000000 from previous operation, if rising edge of the clock signal arrives, output becomes IR+ = 1000000010000000 (0x8080) which means Load LSB operation was done.

(iii) When E = 1, FunSel = 00 and IR = 1000000010000000 from previous operation, if rising edge of the clock signal arrives, output becomes IR+ = 1000000001111111 (0x807F) which means decrement operation was done regardless of L/H.

(iv) When E = 1, FunSel = 01 and IR = 1000000001111111 from previous operation, if rising edge of the clock signal arrives, output becomes IR+ = 100000001000000 (0x8080) which means increment operation was done regardless of L/H.

(v) When E = 1, FunSel = 11, if rising edge of the clock signal arrives, output becomes IR+ = 0000000000000000 (0x0) which means clear operation was done regardless of L/H.

(vi) When E = 0 and IR = 0000000000000000 from previous operation, if rising edge of the clock signal arrives, out becomes IR+ = 0000000000000000 which is the value of IR regardless of FunSel and L/H.

3.2 Part 2

3.2.1 a)

(i) When RegSel = 0000 and TemSel = 0000, all the general and temporary registers become enable. Let the Input = 01100110 (x66) and FunSel = 10. At the rising edge of the clock signal, Input is loaded to all registers. Any combination of the OutASel and OutBSel give the same results which is OutB = OutA = 01100110 (x66).

(ii) (Continue from the state (i)) let FunSel = 01 (increment) and Regsel = TemSel = 0111. At the rising edge od the clock the values stored in R1 and T1 are incremented by one. If OutASel = 000 or OutASel = 100 and OutBSel = 000 or OutBSel = 100, then OutA = OutB = 01100111 (x67). For any other combinations of OutASel and OutBSel, OutA = OutB = 01100110 (x66).

(iii) (Continue from state (ii)) let RegSel = TemSel = 1111. In this situation all the registers become disabled. Any FunSel combination will not effect the stored values. OutA and OutB selection combinations are exactly the same as the state (ii).

3.2.2 b)

(i) When RegSel = 000, all the address register become enable. Let FunSel = 10 and Input = 01000011 (x43). In this situation Input value will be loaded into all registers at the rising edge of the clock signal and any combination of OutCSel and OutBSel will give the same result, OutC = OutB = 01000011 (x43).

(ii) (continue from the state (i)) let RegSel = 011 and FunSel = 11. In this situation only enabled register is PC and clear operation is done at the rising edge of the clock signal. If the multiplexers selects the PC (00 or 01) outputs shows 00000000 (x00). Any other combinations of OutCSel and OutDSel shows the stored value in the previous state which is 01000011 (x43).

3.3 Part 3

3.3.1 For FunSel = 0100 - Addition

(i) When A = 00010111 (0x17) and B = 01100100 (0x64), the output becomes 01111011 (0x7B) which verified the addition operations (23 + 100 = 123). Any flag case (ZCNO) outputs arised.

(ii) When A = 01010111 (0x57) and B = 11100100 (0xE4), the output becomes 00111011 (0x3B). Also, carry flag is equal to 1 at the rising edge of clock signal since there is carry output. It verified the addition (87 + 228 = 256 + 59, carry: 1 (256)).

3.3.2 For FunSel = 0101 - Addition with Carry

(i) When A = 11010111 (0xD7) and B = 01100100 (0X64), output of carry flag is 1 and the ALU output becomes 00111100 (0x3C) which verified the A + B + carry (-41 + 100 + 1(carry) = 60).

(ii) When A = 01111111 (0x7F) and B = 01111111 (0x7F), the output becomes 11111110 (0xFE). There is an overflow case and the results is negative, so 'N' and 'O' from ZCNO output are equal to 1. It is clear that the ALU works correctly in this case (127 + 127 = -2).

3.3.3 For FunSel = 0110 - Subtraction

(i) When A = 00010101 (0x15) and B = 01010101 (0x55), the output becomes 11000000 (0xC0) and the flag 'N' is equal to 1 at the rising edge of the clock signal since the output is negative (21 - 85 = -64).
When A = 00101101 (0x2D) and B = 00101101 (0x2D), the output becomes 00000000 (0x00). The flag 'Z' is equal to 1 at the rising edge of the clock signal since the result is 0 (45 - 45 = 0).

(ii) When A = 10010111 (0x97) and B = 01000111 (0x47), the output becomes 01010000 (0x50). There is an overflow case since the result exceed the limit -127. Also there is a carry. So, 'C' and 'O' ZCNO output are equal to 1 at the rising edge of the clock signal. We can clearly see overflow looking the output ((-105) - 71 = 80). OutALU and ZCNO output give correct result.

3.3.4 For FunSel = 0111 - AND

(i) When A = 11001110 (0xCE) and B = 11110111 (0xF7), the output becomes 11000110 (0xC6). Output 'N' is equal to 1 at the rising edge of the clock signal since last bit of output is 1 and it represent a negative number.

(ii) When A = 10101010 (0xAA) and B = 01010101 (0x55), the output becomes 00000000 and output 'Z' is equal to 1 since the output is zero. It is clear that the outputs give results of A & B, These cases verified that AND operation works correctly.

3.3.5 For FunSel = 1000 - OR

(i) When A = 11001110 (0xCE) and B = 00110111 (0x37), the output becomes 11111111 (0xFF). Output 'N' is equal to 1 at the rising edge of the clock signal since last bit of output is 1.

(ii) When A = 00000111 (0x07) and B = 11100000 (0xE0), the output becomes 11100111 (0xE7). It can be seen that the outputs give results of A || B and OR operation works right.

3.3.6 For FunSel = 1001 - XOR

(i) When A = 11111111 (0xFF) and B = 00000000 (0x00), the output becomes 11111111 (0xFF). Output 'N' is equal to 1 at the rising edge of the clock signal since most significant bit of output is equal to 1 which means that it is negative number.

(ii) When A = 00110101 (0x35) and B = 00110101 (0x35), the output becomes 00000000 (0x00). Output 'Z' becomes 1 at the rising edge of the clock signal because the output is equal to zero. These cases verified that XOR operation works correctly.

3.3.7 For FunSel = 1010 - LSL A

(i) When A = 10110001 (0xB1), the output becomes 01100010 (0x62), and carry output 'C' is equal to 1 since most significant bit(A_7) of A is 1. It shifted the bits to the left correctly, so it's verified.

3.3.8 For FunSel = 1011 - LSR A

(i) When A = 10110001 (0xB1), the output becomes 01011000 (0x58), and carry output 'C' is equal to 1 since least significant bit(A_0) of A is 1. It shifted the bits to the right correctly, so it's verified.

3.3.9 For FunSel = 1100 - ASL A

(i) When A = 10110001 (0xB1), the output becomes 01100010 (0x62). It is similar to LSL A, the only difference is that the arithmetic shift left does not have the carry output as opposed to the logical shift left. According to the output, it is verified that the ASL operation works correctly.

3.3.10 For FunSel = 1101 - ASR A

(i) When A = 10110001 (0xB1), the output becomes 11011000 (0xD8). It shifted arithmetically right and the output gave correct result.

3.3.11 For FunSel = 1110 - CSL A

(i) When A = 11100110 (0xE6), the output becomes 11001101 (0xCD)(least significant bit value takes most significant bit value). Output 'N' becomes 1 since most significant bit of result is 1 and output 'C' also becomes 1 since carry flag(most significant bit of A) is equal to 1. A shifted circular left correctly, so it is verified that CSL works.

3.3.12 For FunSel = 1111 - CSR A

(i) When A = 10101110 (0xAE), the output becomes 01010111 (0x57). The carry output is 0. It shifted circular right correctly, so it is verified that CSR works.

3.4 Part 4

- Case-1

(i) Between T_0 and T_7, values 1,2,3,4,5,6,7 and 8 are written into R1, R2, R3, R4, T1, T2, T3 and T4, respectively.
All registers in the address register file are disabled. IR is disabled.
FunSel of general register file = 01.
If the corresponding register reaches the value aimed, then disable it.

(ii) $T_8 : M[AR] \leftarrow R4 + T4, Ar \leftarrow AR + 1$
Write R4 + T4 into memory. OutASel = 011, OutBSel = 111. All the registers in the general register file are disabled. AR is enabled with FunSel = 01. OutDSel = 10. Alu FunSel = 0100.
Expected Result: M[00] = 12 (x0C) and ZCNO = 0000

(iii) $T_9 : M[AR] \leftarrow R3 - T3, AR \leftarrow AR + 1$
Write R3 - T3 into memory. OutASel = 010, OutBSel = 110. ALU FunSel = 0110. AR FunSel = 01. OutDSel = 10.
Expected Result: M[01] = -4 (xFC), ZCNO = 0010 (C = 0 since there is a borrow).

(iv) $T_{10} : M[AR] \leftarrow R2|T2, AR \leftarrow AR + 1$
Write R2 — T2 into memory. OutASel = 001, OutBSel = 101. ALU FunSel = 1000. AR FunSel = 01. OutDSel = 10.
Expected Result: M[02] = 6 (x06) and ZCNO = 0000.

(v) $T_{11} : M[AR] \leftarrow LSL(T4), T4 \leftarrow LSL(T4), AR \leftarrow AR + 1$
Write Logical Left Shifted value of T4 into memory and T4. OutASel = 111, OutBSel = xxx. ALU FunSel = 1010. Enable T4 (TemSel = 1110) with Funsel = 10. AR FunSel = 01. OutDSel = 10. Expected Result: M[03] = 16 (x10) and ZCNO = 0000.

(vi) $T_{12} : M[AR] \leftarrow LSL(T4), T4 \leftarrow LSL(T4), AR \leftarrow AR + 1$
Properties are same with (v)
Expected Result: M[04] = 32 (x20) and ZCNO = 0000.

(vii) $T_{13} : M[AR] \leftarrow LSL(T4), T4 \leftarrow LSL(T4), AR \leftarrow AR + 1$
Properties are same with (v)
Expected Result: M[05] = 64 (x40) and ZCNO = 0000.

(viii) $T_{12} : M[AR] \leftarrow LSL(T4), T4 \leftarrow LSL(T4), AR \leftarrow AR + 1$
Properties are same with (v)
Expected Result: M[06] = -128 (x80) and ZCNO = 0010.

(ix) $T_{13} : IR(0-7) \leftarrow M[PC], PC \leftarrow PC + 1$
Fetch instruction. Disable AR, Enable PC(initially 0) with FunSel = 01. Enable IR with FunSel = 10 and L/$\sim$H = 1. All the registers in the general

register file are disabled. OutDSel = 00.
Expected Result: IR(0-7) = 12 (x0C).

(x) $T_{14} : R4 \leftarrow IR(0-7)$
Write IR into R4. All the register except R4 are disabled. MuxASel = 10. R4 FunSel = 10.
Expected Result: R4 = 12 (x0C).

(xi) $T_{15} : M[AR] \leftarrow R4 + T4$
Write R4 + T4 into memory. OutASel = 011, OutBSel = 111. All the registers in the general register file are disabled. OutDSel = 10. Alu FunSel = 0100.
Expected Result: M[07]= -116 (x8C)and ZCNO = 0010.

(xii) $T_{16} : IR(8-15) \leftarrow M[AR]$
Write M[AR] into IR(8-15). OutDSel = 10. L/∼H = 0.
Expected Result: IR(8-15) = -116 (x8C).

- Case-2

(i) $T_0 : T1 \leftarrow T1 + 1$
Increment T1. Inputs of Register File OutASel = 100, OutBSel = 000, FunSel = 01, RegSel = 1111, TmpSel = 0111; inputs of RAM as str = 0, sel = 0, ld = 0, clr = 0 and Enable = 0 for Instruction Register. In this way, T1 register's data is written to OutA and R1 register's data is written to OutB. Initially all the register's data are 0 so that OutA and OutB are 0. Increment operation is selected by FunSel.
Expected Result: T1 = 1.

(ii) $T_1 : T1 \leftarrow T1 + 1$
All the properties are same as (i).
Expected Result: T1 = 2.

(iii) $T_2 : T1 \leftarrow T1 + 1$
All the properties are same as (i).
Expected Result: T1 = 3.

(iv) $T_3 : M[AR] \leftarrow CSR(T1), AR \leftarrow AR + 1$
Write circular right shifted value of T1 into memory. TmpSel is switched as TmpSel = 1111, now OutA = 00000011 (3). FunSel of ALU is switched as FunSel = 1111 which corresponds to Circular Shift Right operation for input A (00000011). Inputs of Address Register File is switched as OutDSel = 10, FunSel = 01 and RegSel = 101. In this way, AR register's data(0) is written to output so that address 0 is selected in Memory and according to FunSel data

in AR will be incremented by 1 at the rising edge of the clock cycle. Inputs of Memory is switched as str = 1 and sel = 1 so that the data will be written to selected address(0) at the rising edge of the clock cycle. Expected Result: M[00] = 10000001 (x81) and ZCNO = 0110.

(v) $T_4 : M[AR] \leftarrow CSL(T1)$
Write circular left shifted value of T1 into memory. FunSel of ALU is switched as FunSel = 1110 which corresponds to Circular Shift Left operation for input A (00000011). Address Register File's Regsel is switched as RegSel = 111 so that AR's data will not change at the rising edge of the clock cycle. Now AR register's data(1) is written to output so that address 1 is selected in Memory. Inputs of Memory is switched as ld = 1 so that the data will be written to selected address(1) and data in address 1 will be read to Output of Memory at the rising edge of the clock cycle.
Expected Result: M[01] = 00000110 and ZCNO = 0000.

(vi) $T_5 : I(8-15) \leftarrow M[AR]$
Memory's str is switched as str = 0 to prevent changes in address and data in memory. Instruction Register's inputs are set as L/H = 0, FunSel = 10 for load operation and Enable = 1. Now Instruction Register's I is I = 00000110 which is coming from Memory.
Expected Result: IR(8-15) = 00000110(0x06).

4 DISCUSSION

4.1 Part 1

In this part, we have designed two different types of registers. One of them is 8-bit the other one is 16-bit registers. We know that registers consist of flip-flops so both registers which we designed consist of D-flip flops to perform their functionality in a synchronised way according to clock signal.

The 8-bit register takes 8-bit data as input and then it gives 8-bit data as output. It takes also two inputs which are 2-bit FunSel and 1-bit E both to regulate functionality. According to FunSel input, register decides to which operation will be performed when the rising edge of the clock cycle arrives. E input decides whether the register is enabled or not which means that whether the register can perform its chosen functionality or not. It is an undeniable fact that 8-bit register which has ability to perform elementary operations is the fundamental unit of our overall design.

The 16-bit register which is designed as a Instruction Register (IR) takes 8-bit data as input and then it gives 16-bit data as output. It takes also three inputs which are 1-bit L/H', 2-bit FunSel and 1-bit E all three to regulate functionality. According to FunSel input, register decides to which operation will be performed when the rising edge of the clock cycle arrives. E input decides whether the register is enabled or not which means that whether the register can perform its chosen functionality or not. L/H' input chooses the half of the storage to write while loading operation is selected by FunSel. It can be claimed that we need IR so much for our design.

4.2 Part 2

In this part, we designed two register files. First one is General Register File which consists of 8 different registers (designed in Part 1). Second one is Address Register File which consists of 3 different registers (designed in Part 1). Their working behaviours are exactly the same but they are used for different purposes in the whole system. Purpose of the General Register File is to determine the operands of the ALU. On the other hand, the purpose of the Address Register file is to determine which chip of the memory module will be selected. This decision processes are handled by the 2 multiplexers in the register files.

General Register File can store 8 distinct operand for ALU in the whole system which is very useful. Address Register File stores 3 distinct address values which are PC, AR and SP. As their names suggest we can use PC to determine the address of the next program to execute. We can use AR to use the value as operand or give as output. Finally, we can use SP to keep track of the last programs requested.

This part was easy to design and test. As can be seen in the "results", we have tested with some input combinations and we saw that our design works without any mistake.

4.3 Part 3

In this part, we designed an Arithmetic Logic Unit (ALU). It is important part of the Central Processing Unit (CPU). It handles all the calculations which the CPU needs. In other words, ALU is the "calculator" section of the computer which means that it performs all arithmetic and logic operations. These operations are addition, subtraction,

NOT, AND, OR, XOR gates, left and right shift operations (logical/arithmetic/circular). In our design, we have 16 different function and which operation is to be performed is chosen thanks to FunSel selection input. We divide processes into six parts which are arithmetic unit, logic unit, logical shift unit, arithmetic left shift unit, arithmetic right shift unit, circular shift unit and the others (NOT gate and load are used directly).

This part took a lot of our time because we designed each unit module individually. After implementation, we tested some cases for each function one by one to check if the ALU system is working or not. According to the results, we are convinced that it works correctly. We also checked for zero, carry, negative, overflow conditions for each function. In particular, we tried to select and show where these conditions (ZCNO) occur.

4.4 Part 4

In this part, we completed the small processor system which collaborates with a memory module. Although we completed the circuit, we could not understand how the system works because there are lots of components in the whole system. When we started to play with the circuit we started to understand how the components interact with each other. As can be seen in the results part, we tested our circuit with long test cases. Results of the cases are same with the expected results, we can say that our circuit works properly. First of all, we thought that origin of the circuit is the memory. Hence, memory is initially empty. So, by using ALU and general register file we filled some part of the memory. After that, using data with the different multiplexer selector combinations we tested other functionalities of the system in the cases and our design satisfied all the expected results.

In this part, we realized that the thinking as a whole in the computer organization is very important to understand how small components come together and form a system. We understand that we leveled up from digital circuits to computer organization thanks to this small processor system.

5 CONCLUSION

In this project we had a chance to apply what we learned in the lessons and we refreshed our knowledge about digital circuits. During the experiment, we did not encounter any difficulties worth mentioning. We enjoyed every second of doing the project.