ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 3

DUE DATE : 03.06.2020

GROUP NO : G29

GROUP MEMBERS:

150170054 : ZAFER YILDIZ

150180080 : BAŞAR DEMİR

150180012 : MUHAMMED SALİH YILDIZ

150160802 : MEHMET CAN GÜN

Contents

1 INTRODUCTION

In this project, we designed a basic CPU with some particular Opcodes that specifies the operation which is performed by the circuit we designed. To implement the functionalities of the circuit, as a first step, we converted every opcode to meaningful subcircuits to determine values of input fields of the circuit we designed in Project-2. Then, we read the instructions from the memory and run the circuit successfully.

2 PROJECT PARTS

2.1 Main Part

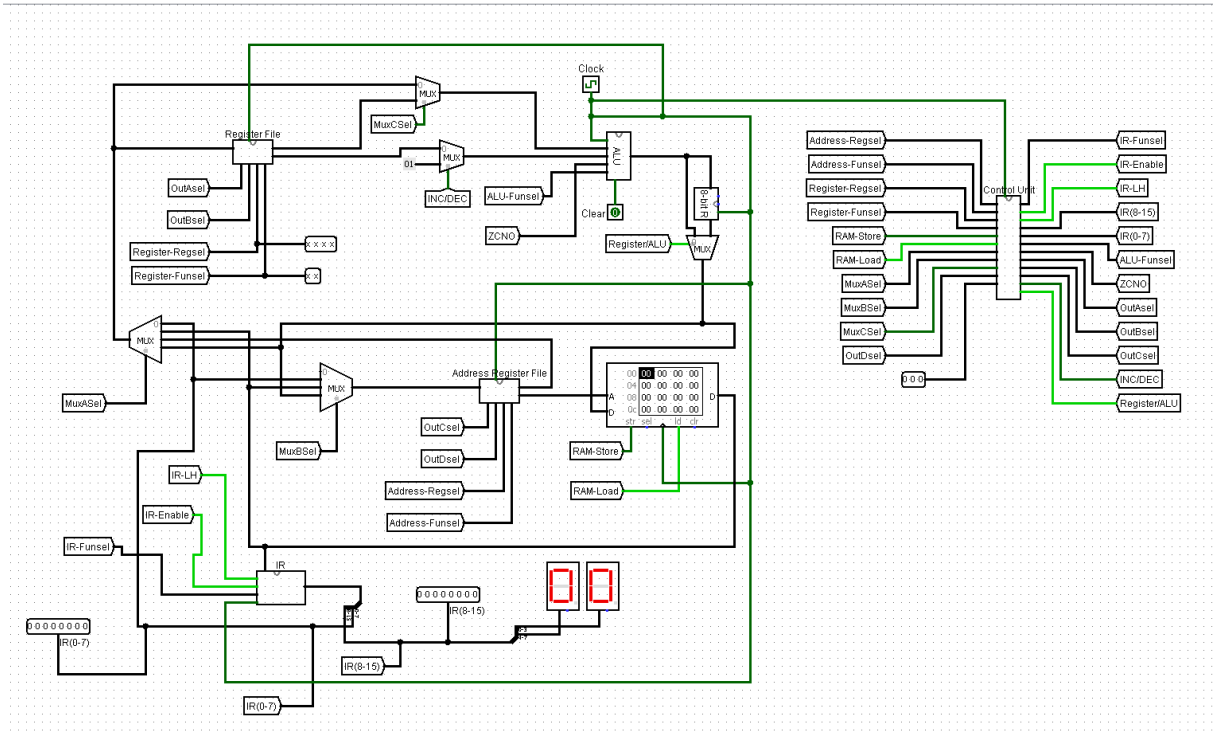

Figure 1: Project-2 and control unit

In project-2 we have designed an ALU and implemented a structure that contains ALU, RAM, Registers File, Address File and Instruction Registers. Also we used several types of multiplexers and inputs to control them . They were designed with manually controlled inputs.

In this project we implemented a control unit to control all inputs of the structure. An instruction that read from RAM, is stored in IR registers and dispatched to the control

unit. Instruction is processed in the control unit to determine inputs of this structure which leads to give expected outputs.

2.2 Timing Issues

Basic computer systems contains lots of synchronous operations.Therefore, every component of the computer must work in a harmony. It is provided by using same clock for whole system.

In our implementation, every part communicates with the clock. They are able to reset or take time information from it.

We have done some analysis about how many clock cycles needed for every operation. Based on this, we constructed clock system with using 3-bit counter. By decoding the output of the counter, we get current time. Our clock implementation is given below.

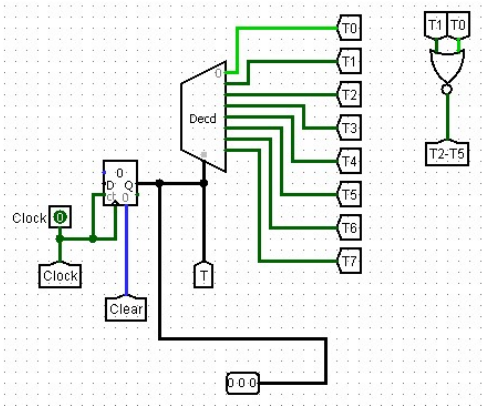

Figure 2: Clock

For every instruction of computer, operations in T_0 and T_1 must be same. In this time units, computer should read instructions from memory, write them to IR and decode them. Therefore, we have implemented circuits for T_0 and T_1. For bigger times, we enable one of the our opcode circuits.

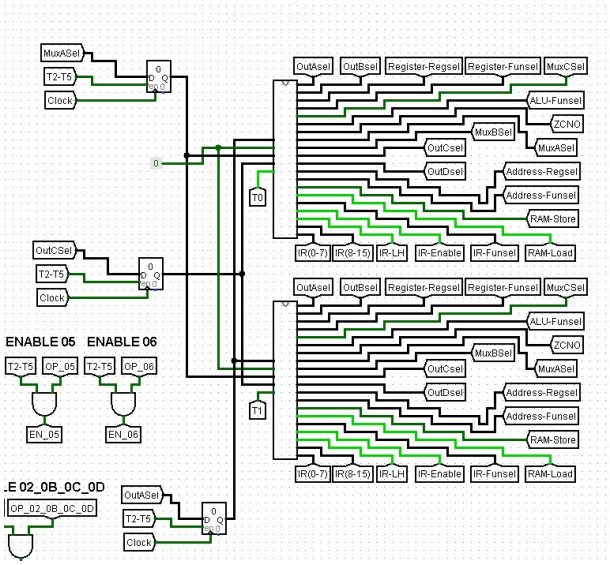

Figure 3: T0 and T1 Circuits

This circuits contain input configurations for removing data from RAM to IR. Related operations are

$$IR[8-15] \leftarrow M[PC], PC \leftarrow PC+1$$

$$IR[0-7] \leftarrow M[PC], PC \leftarrow PC+1$$

While performing these operations, we should keep ALU's output value to get correct results when instruction depends previous instruction. To keep these output values, we have added 3 D-flip flops which store last select bit of previous instruction. By doing this, we can keep ALU's output by giving same result to inside.
Internal structure of T_0 and T_1 circuits are given below.

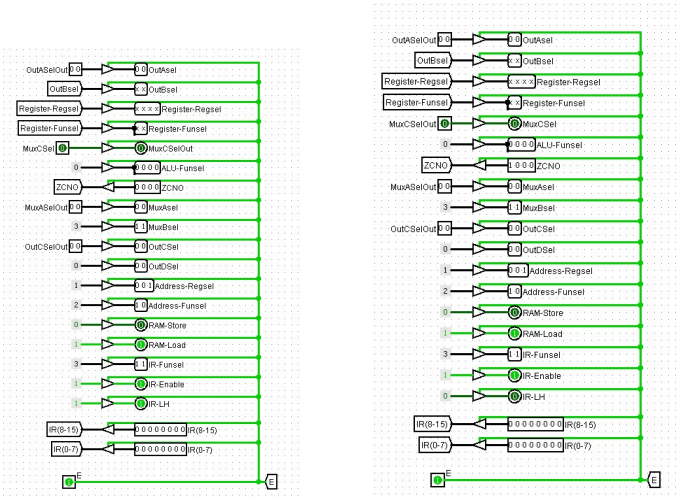

Figure 4: Internal structure of T_0 and T_1

2.3 Opcode Selection

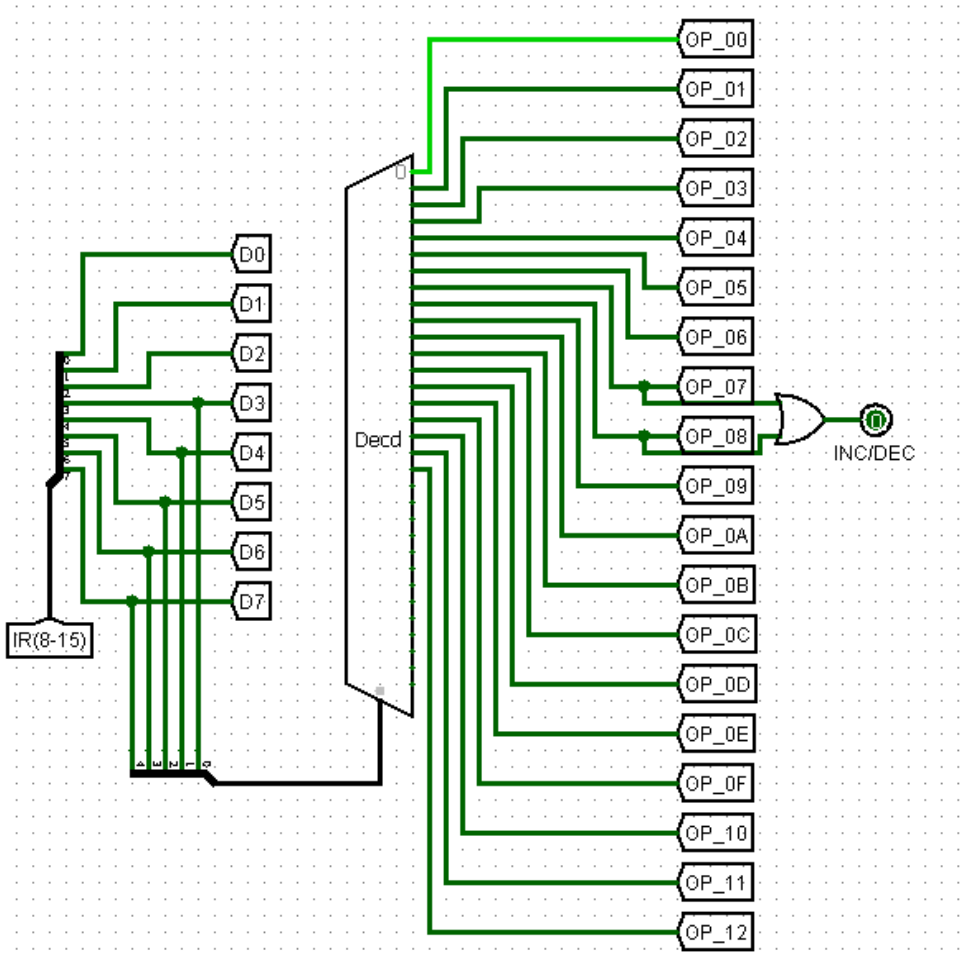

Figure 5: Opcode Decoder

To select corresponding opcode circuit which can perform the given instruction, we used an 5:32 Decoder. Select bits of the decoder are the 5 most significant bits of the 16-bit instruction code.

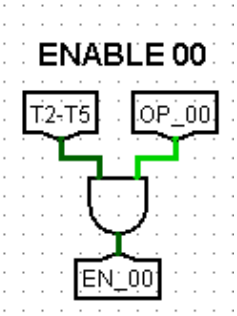

Figure 6: Opcode Enable

We had also enable inputs for every opcode circuit to prevent that every opcode run

at the same time. To determine the enable input we connected the bit that specifies to which opcode will be run and the time range of this opcode to 2-input AND gate.

2.4 Opcode Circuits

We had two different type opcode circuit in this project. One of them is the opcode circuit that performs only one instruction type when other type is the opcode circuit that performs multiple instructions.

2.4.1 Opcode Circuits with only 1 Instruction Type

In this kind of opcode circuits, we decided the inputs of the circuit in Project-2 to perform the given instruction for this opcode circuit. For instance, above circuit is for the BNE operation (0x10). The BNE operation takes only one clock cycle which is T_2 for our design. So, we used controlled buffers to be sure that the opcode circuit runs only in T_2 time range. Then, we gave 0 to unnecessary inputs and we gave suitable values to necessary inputs to perform this operation.

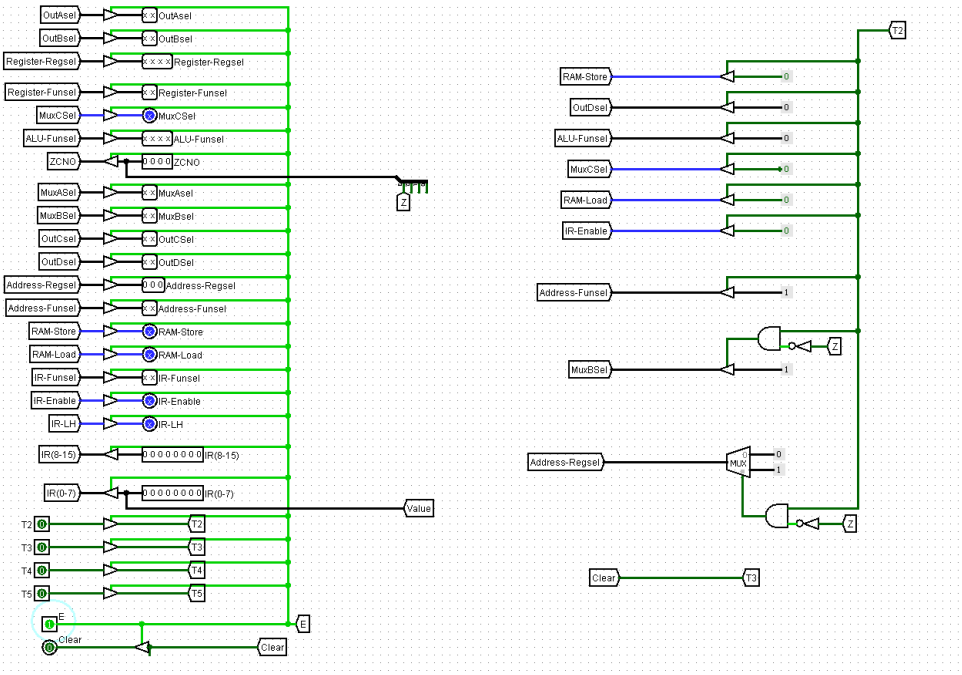

Figure 7: An opcode circuit with only 1 instruction

For instance, the definition of the BNE operation is that if Z flag is 0, then slide Value to PC Register. So, we connected inverse of the Z flag and time bit as select bit to a 2:1 MUX to determine the Address-Regsel input. If the Z flag is 1, Address-Regsel will be 000 to protect the value of the PC register, because the condition is not satisfied in this situation. On the other hand, if Z flag is 0, then Address-Regsel will be 001 to enable

PC Register and Address-Funsel will be 01 for the load operation. And for the MuxBSel, if the Z flag condition is satisfied, the opcode circuit gives 01 to select bits of the MuxB to allow that the value that is loaded to PC register can go into Address Register File. Finally, we gave a 1-bit output which is labelled as Clear in the above circuit to determine when the time is set to 0.

2.4.2 Opcode Circuits with Multiple Instruction Types

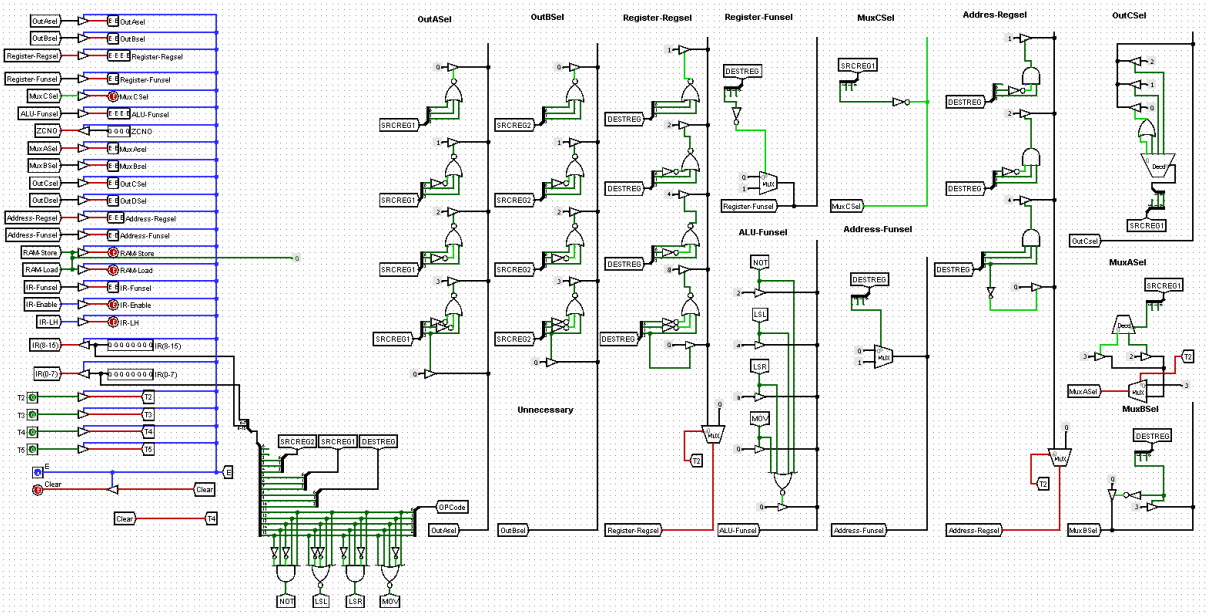

Figure 8: An opcode circuit with multiple tasks

We designed some opcode circuits to do multiple tasks in one circuit. The opcodes that have similar operations are gathered in these circuits.For example in Figure 8 we implemented NOT, LSL, LSR and MOV operations in one circuit. Because all of these operations have nearly the same process except ALU Funsel. In the left most side, inputs and outputs of the circuit are located. First of all DESTREG, SRCREG1, SRCREG2 and OpCode derived from the instruction. OutASel's value is determined according to SRCREG1. Rx's value is sent to the MuxC. OutBSel is unnecessary in this circuit because ALU will do its operations with only one input which comes from input A(MuxC). Register Regsel is determined according to DESTREG because Regsel's are used to load some values to registers. Register Funsel is 01 to choose load operation in Register File or Address File. ALU Funsel is main difference between opcodes. It is determined according to opcode value derived from the instruction and it does main operation of opcode. MuxCSel is 0 if the SRCREG1 is one of the Address Registers else it takes 1 value to get Register Files value that selected by OutASel. Address Funsel is almost the same

with Register Funsel. It is takes 01 value to load to registers. Also Address Regsel is the same with Register Regsel. It determines which register will be loaded with the result. OutCSel is determines which output will be given from Address Registers. It depends on SRCREG1. MuxASel is quite different. It chooses the value comes from ALU if SRCREG1 is from Register File else it chooses the value comes from Address Registers in T_2. But in T_3 it always chooses the value comes from ALU because we need processed result to load Register Files. MuxBSel determines to value comes from ALU will reach to Address Register or not. When all of this processes are finished the circuit send the results to main circuit and resets time to let system to read next instruction.

2.5 Extra Features

We had some problems during this project, so we added some extra features to our circuits to overcome these problems:

One of the problems we faced with is that in INC and DEC operations if we perform increment or decrement operation with FunSel of the register we noticed that Z flag is not updated, because the result did not pass through ALU. The other problem with these two operations is that if we did not perform these operation through ALU, we could not protect the value of the register which is operand in this operation. As a solution, we decided that we can add a MUX to the cable which is between OutB and ALU. And if the operation is INC or DEC, 1 is selected from this MUX and funsel of the ALU is selected as addition or subtraction to perform INC and DEC operations in a successful way. In other operations, value of the OutB does not change.

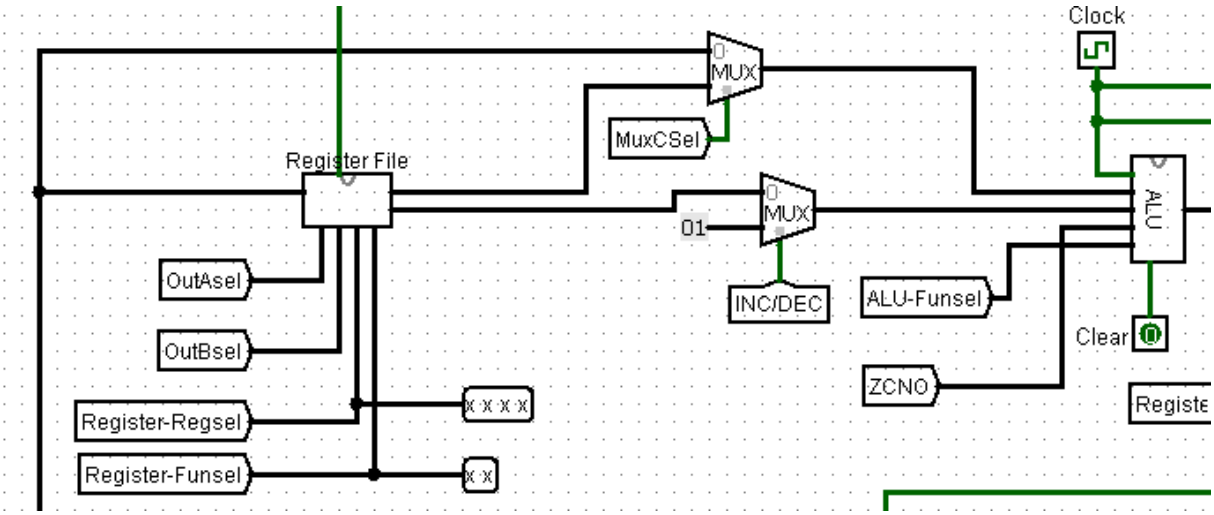

Figure 9: Solution for the INC-DEC operations

Another problem we faced is that, ZCNO flags of our ALU circuit is not updated correctly, because in our ALU circuit we designed in Project-2, ZCNO flags are updated with clock but the result does not depend on the clock. So, in operations which needs to value of the Z flag (BEQ and BNE operations) wrong value of the Z flag is pulled by the corresponding opcode circuit. As a solution, we added an 8-bit register to the output of the our ALU to keep the old result for the updating process of the ZCNO flags in only operations which needs to ALU.

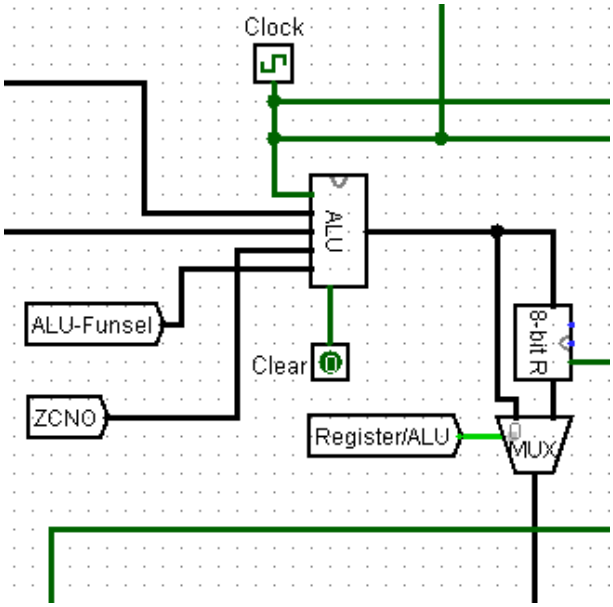

Figure 10: Solution for the ZCNO flags

To determine the operations that needs the ALU in control unit, we connected these opcodes to an OR gate.

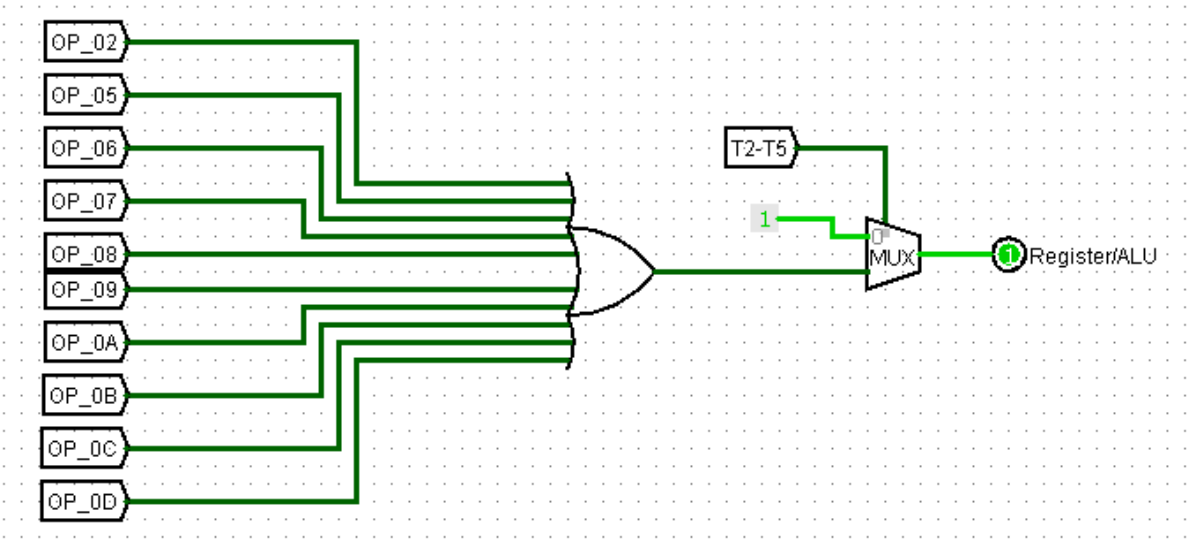

Figure 11: Selection of ALU operations

3 RESULTS

In the result part, we tried different inputs and get expected results for each opcode. In order to prove the purpose of the circuit, we decided to implement the First 9 number of the Fibonacci Series with backwards.

The first part of the code in Table 1 calculates Fibonacci numbers and writes starting from 0xA0 to 0xA8.Then, the codes of the second part as can be seen in Table 2 is called by using the CALL 0x50 command. Also, there are some test cases in order to check the program is produced wrong data or not, which is adding time complexity.

The second part is mostly dealt with stack operations. Firstly, read data from 0xA0 to 0xA8 and push to stack. After the loop is finished, the program is pulled the data from the stack and print from 0xB0 to 0xB8 within backwards. In the end, return the next command.

Table 1: Finding the First 9 Fibonacci Numbers - The First Part from 0x00

	70	02	Start from 0x02
	00	a0	Load A0 to R0
	16	00	Move R0 to AR
	00	00	Load 0 to R0
	46	c0	Increment AR by 1
	02	01	Load 1 to R1
	0a	00	Store R1 to M[AR]
	2a	04	Sum R0 and R1 to R2
	46	c0	Increment AR by 1
	0d	00	Store R2 to M[AR]
	10	20	Move R1 to R0
	11	40	Move R2 to R1
	06	ff	Load FF to R3
LABEL1:	2a	04	Sum R0 and R1 to R2
	46	c0	Increment AR by 1
	0d	00	Store R2 to M[AR]
	10	20	Move R1 to R0
	11	40	Move R2 to R1
	32	c4	Substract AR and R1 to R2
	33	68	Substract R3 and R2 to R3
	3b	c0	Decrement AR by 1 to R3
	4a	04	AND R0 and R1 to R2
	5a	40	NOT R2 to R2
	53	68	OR R3 and R2 to R3
	6b	60	LSR R3 to R3
	05	00	Load M[AR] to R2
	62	40	LSL R2 to R2
	33	68	Substract R3 and R2 to R3
	63	60	LSL R3 to R3
	6b	60	LSR R3 to R3
	04	07	Load A8 to R2
	13	C0	Move AR to R3
	4b	68	AND R2 and R3 to R3
	80	1a	IF Z=0;GO TO LABEL1
	02	FF	Load FF to R1
	17	20	Move R1 to SP
	88	50	CALL 50

Table 2: The second part from 0x50

	00	A0	Load A0 to R0
	16	00	Move R0 to AR
	04	08	Load 07 to R2
LABEL2:	03	00	Load M[AR] to R1
	19	00	PUSH R1 to Stack
	46	c0	Increment AR by 1
	38	c0	Decrement AR by 1 to R0
	4b	08	AND R0 and R2 to R3
	78	56	IF Z=1; GOTO LABEL2;
	00	B0	Load B0 to R0
	16	00	Move R0 to AR
	00	08	Load 08 to R0
	04	00	Load 00 to R2
LABEL3:	21	00	PULL M[SP] to R1
	0a	00	Store R1 to M[AR]
	46	c0	Increment AR by 1
	42	40	Increment R2 by 1
	4b	08	AND R0 and R2 to R3
	78	6a	IF Z=1; GOTO LABEL3;
	21	00	PULL M[SP] to R1
	0a	00	Store R1 to M[AR]
	90	00	Return next command

The result is shown in Memory as shown in Figure 12.

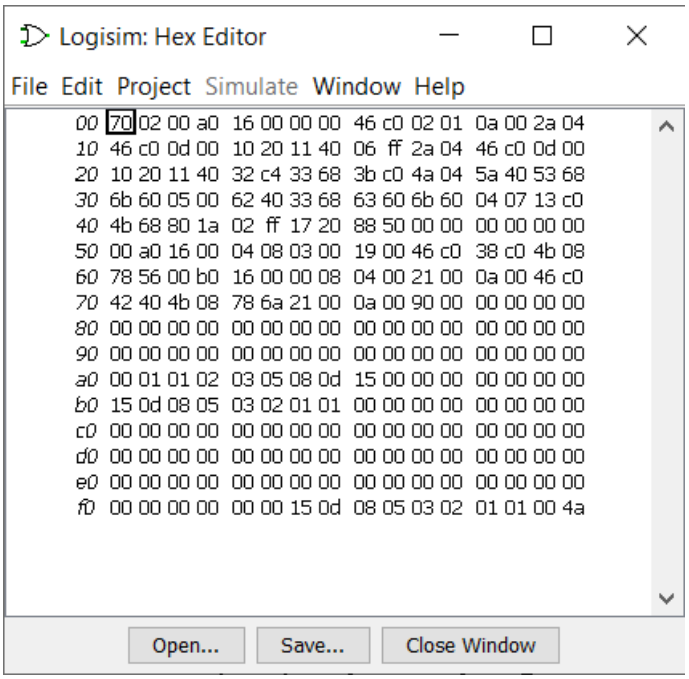

Figure 12: The Logisim Memory Code of the Program

4 DISCUSSION

At the start, we have analyzed the homework and discussed how to implement the circuit. With the help of the presentations, we have learned principles of a basic computer system. We have done examples in group meetings and all of us understood the fundamental architecture of the planned circuit. Logisim is not a great application for group work. Because of this, we divided all opcodes and we constructed separate circuits for all. It provides us easy debugging and fast implementation chance. Because of the higher input combination case, the implementation phase of opcode circuits was hard. We have tested all cases using HW2 and implement proper circuits. It needs a high concentration and good analysis. Then, we have gathered this opcode circuits in a control unit. The control unit consists of timing units and opcodes. They work in harmony and every opcode can decide which time their operation ends and sets the time to T_0. Thus, it provides flexible timing opportunity and higher operating speed.

In testing and optimization part of our homework approximately takes 4 days. We divided opcodes to team members randomly. All of us tested our and randomly distributed opcodes. We debugged problems of circuits and we added new components to HW2 circuit to handle problems. We wrote lots of test cases and we ran these tests in online group meetings. We thought that is very educational homework to reinforce our computer organization knowledge. It was very satisfying to see it works properly.

5 CONCLUSION

In conclusion, we designed a basic CPU circuit that processes given 16-bit instructions in this project by using the circuits we have implemented in previous projects. We had also some problems during the project, but we solved all problems we noticed together with creative solutions. As a final comment, it was a really great experience for us, because we got a chance to try and develop our logic design skills during all these projects from the first one to the last one.