ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 4

DUE DATE : 14.07.2020

GROUP NO : G29

GROUP MEMBERS:

150170054 : ZAFER YILDIZ

150180080 : BAŞAR DEMİR

150180012 : MUHAMMED SALİH YILDIZ

150160802 : MEHMET CAN GÜN

Contents

FRONT COVER

CONTENTS

1 INTRODUCTION **1**

2 PROJECT PARTS **1**

2.1 Main Part 1

2.2 Mapping 2

2.3 Micro-Instruction Format 3

2.4 Control Unit 4

2.4.1 Interpreting IR in Control Unit 4

2.4.2 Main Part of the Control Unit 5

2.4.3 F1-F2 Micro-operations 7

2.4.4 OutSels 8

2.4.5 Regsels and RAM Control 9

3 RESULTS **10**

4 DISCUSSION **13**

5 CONCLUSION **14**

1 INTRODUCTION

In this project, we designed a micro-programmed CPU with some particular Opcodes that specifies the operation which is performed by the circuit we designed in Project-3. To convert the hard-wired circuit we designed in Project-3 to a micro-programmed circuit, we used a mapping technique which specifies the location of the micro-operation that will be run by the main circuit in the ROM that contains micro-instructions for OpCodes.

2 PROJECT PARTS

2.1 Main Part

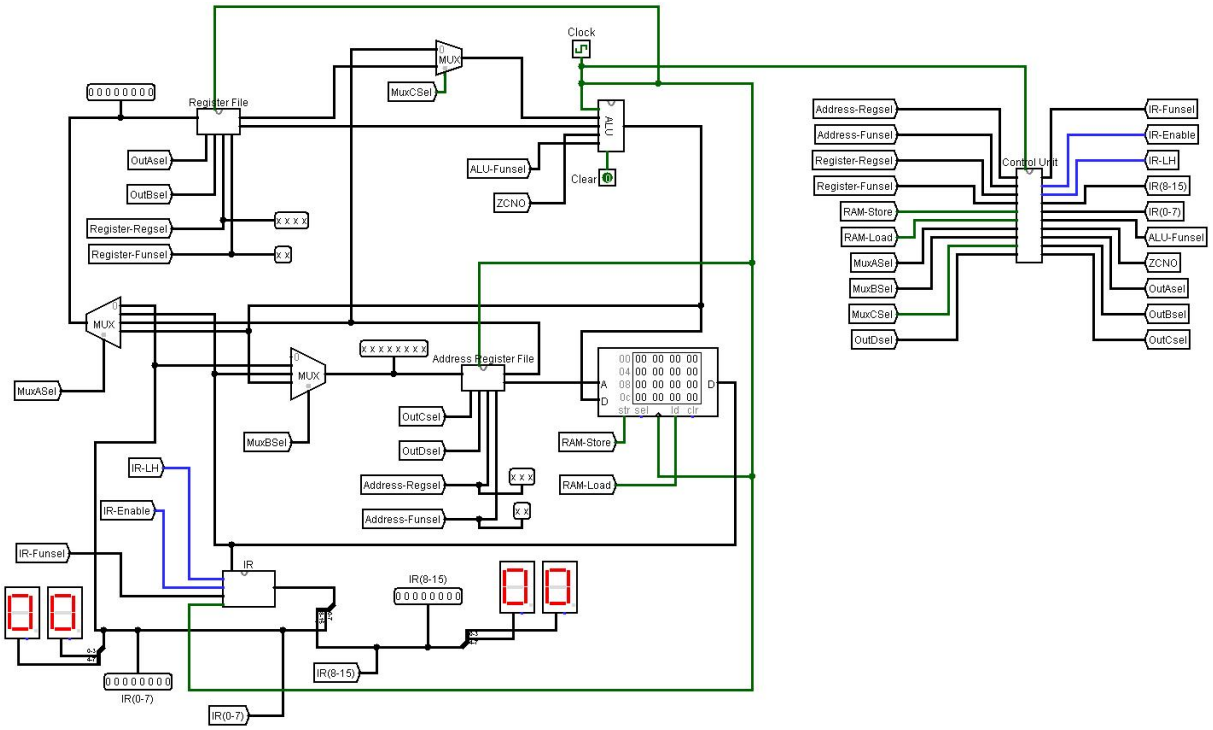

Figure 1: Project-2 and Software Based Control Unit

In project-2, we have designed an ALU and implemented a structure that contains ALU, RAM, Register 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 software based 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 with mapping and interpretation of corresponding micro-instruction to determine inputs of this structure which leads to give expected outputs.

2.2 Mapping

Mapping is basically an addressing mechanism that establishes a connection between instructions (OpCode) and micro-instructions. To build that mechanism, we used a ROM which keeps addresses of micro-instructions.

Firstly, we have calculated number of operations for each opcode, then we have decided to give 4 ROM address for each micro-instruction. Therefore, we have filled our mapping ROM with addresses that have 4 step size.

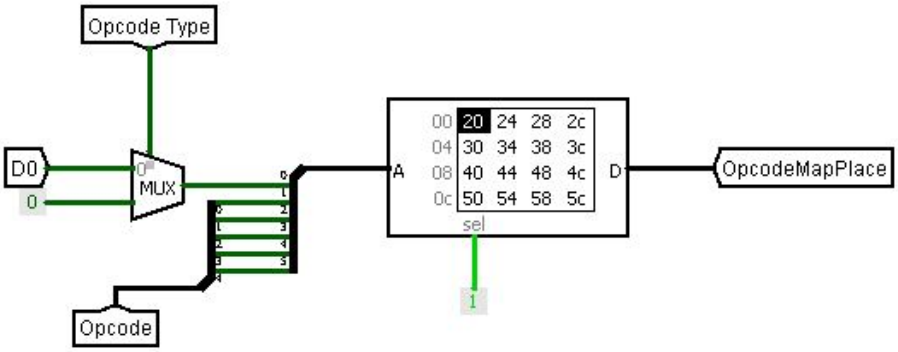

Figure 2: Mapping Mechanism

Address input of this ROM consist of one 6-bit splitter. 1-5 bits directly come from opcode of instruction which is taken from Instruction Register. 0-bit is used to determine addressing mode of instructions with address reference. If addressing mode is immediate and it maps to address X, then if addressing mode is direct, it maps to $X + 1$. Therefore, we can assign different micro-instructions for same opcodes depending on its addressing mode. For instructions without address reference, we do not need any distinction for opcodes, so we give 0 to first bit.

2.3 Micro-Instruction Format

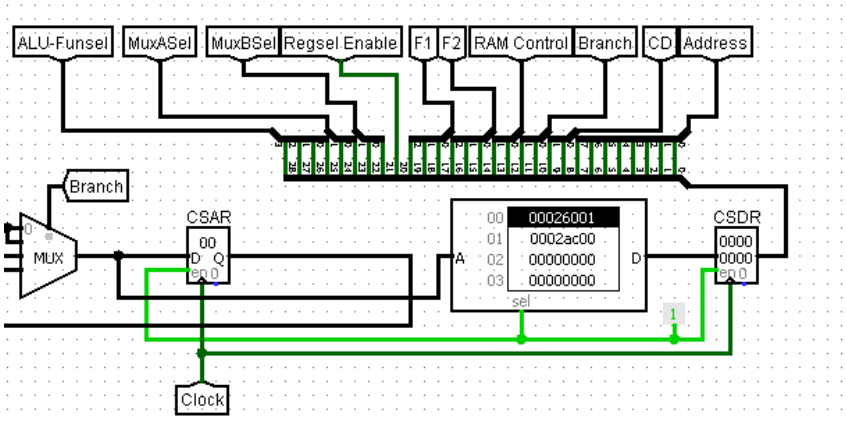

Figure 3: ROM and Micro-instruction

The fundamental structure of the software-based control unit is micro-instruction. Therefore, we have determined the needs of our instructions. Each instruction that is read from RAM contains source, destination or another required information. For this reason, we did not write micro-instruction for each instruction case. We have developed a generalized micro-instruction approach for instructions. We have taken some information directly from RAM. Our 29-bit micro-instruction design is given below.

ALU FunSel	MUX A	MUX B	RegSel Enable	F1	F2	RAM Control	Branch	Condition	Address
4-bit	2-bit	2-bit	1-bit	3-bit	3-bit	2-bit	2-bit	2-bit	8-bit

- ALU Funsel → Directly controls ALU
- MUX A → Directly controls MUX A
- MUX B → Directly controls MUX B
- RegSel Enable → Enables RegSels of registers
- F1 → Contains 7 micro-operation definitions
- F2 → Contains 7 micro-operation definitions
- RAM Control:
  - 00 → Both of load and store of RAM is not enabled.
  - 01 → Only RAM Store is enabled.
  - 10 → Only RAM Load is enabled.

– 11 → Both of load and store of RAM is enabled.

- Branch → It determines which branch mechanism will be performed.
- Condition → It contains condition cases
- Address → It contains branch address

2.4 Control Unit

2.4.1 Interpreting IR in Control Unit

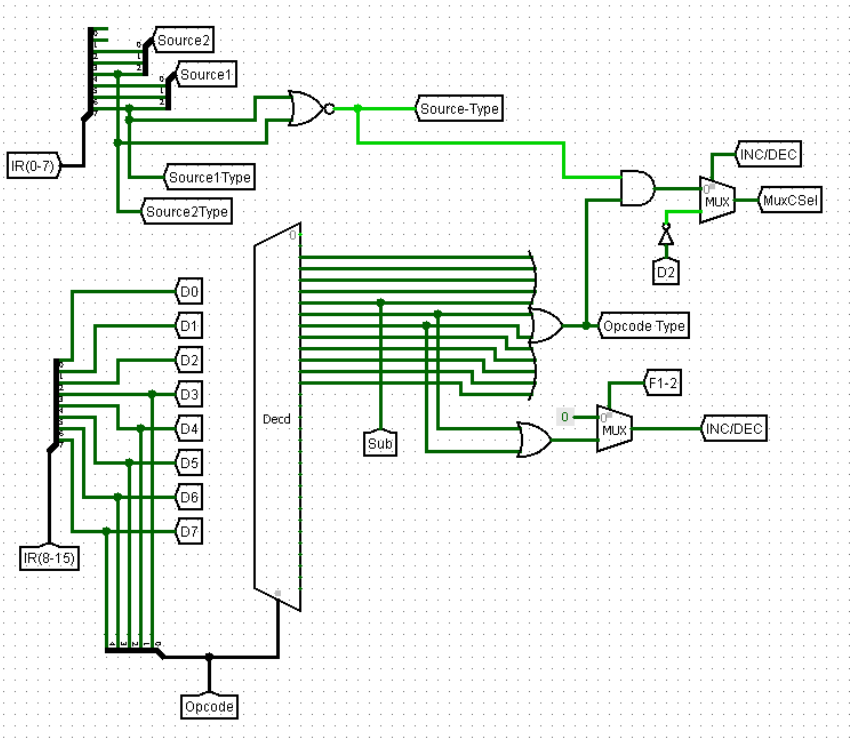

Figure 4: Explanation of IR

In the control unit, we have two micro-instructions that read 16-bits instruction from RAM and load to IR(8-15) and IR(0-7) respectively. The most significant 5 bits of the IR are opcode always. As shown in Figure 4, the opcode is assigned according to IR(8-15). In the case of the opcode is between the interval of $0x02$ and $0x0d$, it refers that the opcode is without reference type, then Opcode Type Flag becomes 1. Also, the SUB, the BNE, and the INC/DEC flags are determined in this part. On the other hand, IR(0-7) part contains the value of Source1 and Source2. Source1Type and Source2Type that will decide to values of the OutSels are also assigned. At the same time, Source-Type, which demonstrates the types of sources, is decided. If both of the sources are register

file registers than Source-Type takes 1 otherwise, it takes 0. The MuxCSel is decided at the end of all these processes. If the operation is not increment or decrement, it becomes output of the AND gate whose inputs are Opcode Type and Source-Type respectively if the operation is increment or decrement, it becomes complement of D_2 which is 11^{th} bit of IR.

2.4.2 Main Part of the Control Unit

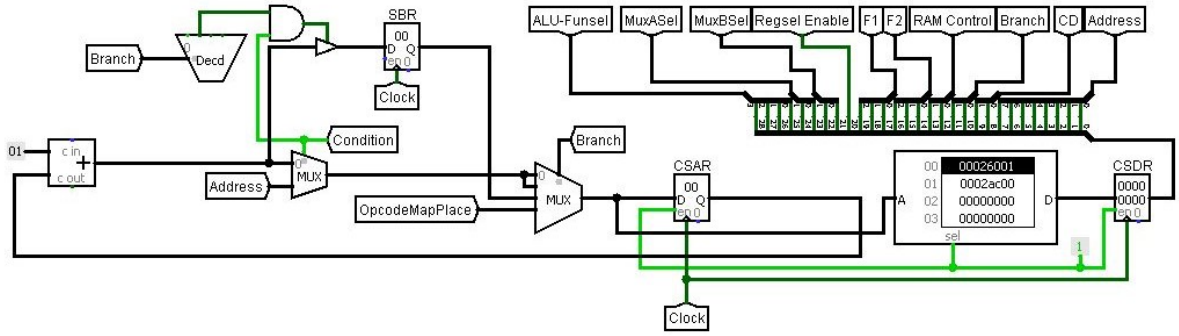

Figure 5: Main Part of the Control Unit

In this part of the circuit, we constructed next-address generation logic based on BRANCH and CONDITION values. We have CSAR to move to the next micro-instruction code in the instruction ROM and the value that will be loaded to this register is selected by 4:1 MUX based on BRANCH value. Here is the table showing the selection of the address value that will be loaded to CSAR:

BRANCH	Function	
00	CAR← CAR+1	if condition=0
	CAR← AD	if condition=1
01	CAR← CAR+1	if condition=0
	CAR← AD, SBR← CAR+1	if condition=1
10	CAR←SBR	
11	Address value from mapping ROM	

When we look at the BRANCH table, we saw that some operations are based on CONDITION value. We constructed another structure to decide the value of condition which is given below:

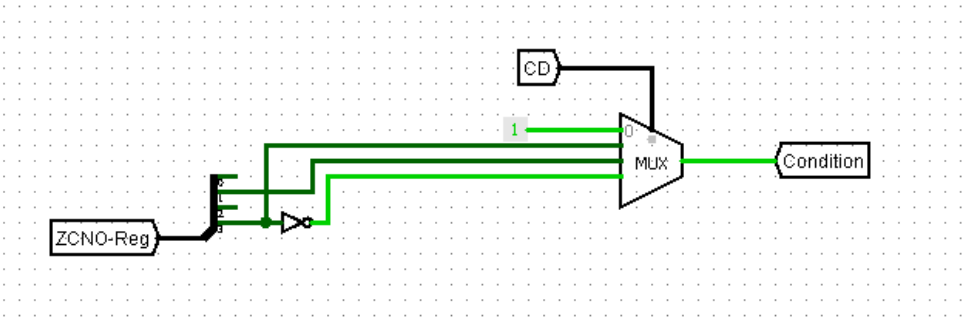

Figure 6: Determination of Condition

If the CD value in the micro-instruction is 00, then Condition will be directly 1 (Unconditional Branch). If the CD is 01, then Condition will be value of the Zero flag of the ALU for BEQ operation. If the CD value is equal to 10, the condition will be value of the Negative Flag of the ALU at this time. Finally, if the CD is equal to 11, then Condition will be complement of the Z flag for the BNE operation. We used 01 and 11 for BEQ and BNE operations that requires Z flag condition. BEQ will be performed if the Z flag is equal to 1, when the BNE will be performed if the Z flag is equal to 0. Here is the table showing determination criteria of the CONDITION value:

CD	Condition	Comments
00	Always=1	Unconditional Branch
01	Z	Zero Flag of the ALU
10	N	Negative flag of ALU
11	$\overline{Z}$	Complement of Zero Flag

If we turn back to the Figure 4, we have a 29-bit micro-instruction ROM that keeps the micro-instructions in specific addresses that can be calculated our mapping technique. Actually we can say that this ROM is the heart of the our micro-programmed control unit, because control inputs is sent by this ROM to all circuit elements. Here is the written micro-instructions in the ROM:

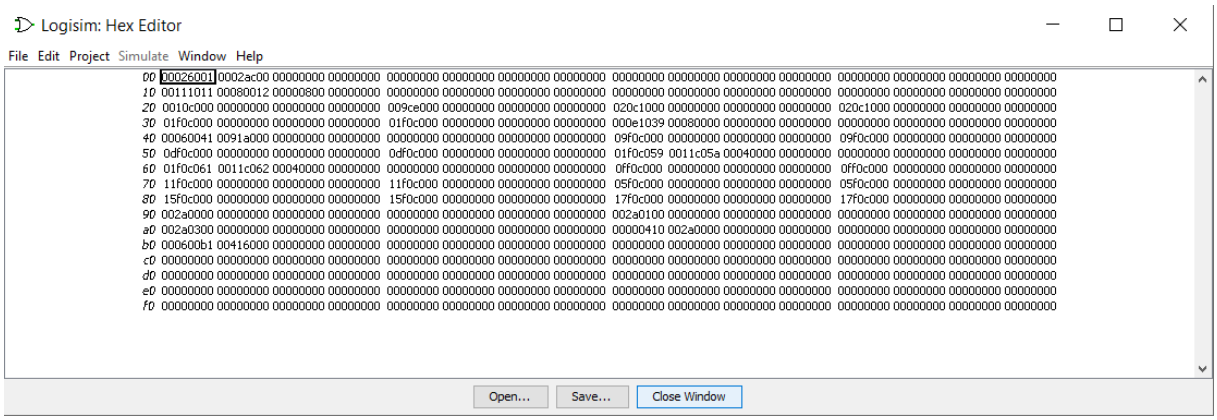

Figure 7: Content of the Instruction ROM

As a final element, we have another register which is loaded by 29-bit micro-instruction value which is written in the instruction ROM. This register which is called CSDR in our circuit keeps the micro-instruction of the operation that will be run by the CPU. We splitted the bits with necessary widths and send these values to necessary locations in the circuit with tunnels.

2.4.3 F1-F2 Micro-operations

We used two 3-bit F values to represent circuit operations to transfer necessary inputs from control unit to main circuit which are F_1 and F_2. Each F has 8 different operations which are given in the table below:

	F1	F2
000	None	None
001	PC+1	IR(L)←M[AR]
010	Pass from ALU	IR(H)←M[AR]
011	SP+1	FunSel Load Operation
100	SP-1	M[SP]←PC
101	PC←Address	PC←M[SP]
110	Load from M[AR]	R_x←M[SP]
111	M[SP]←R_x	INC/DEC

2.4.4 OutSels

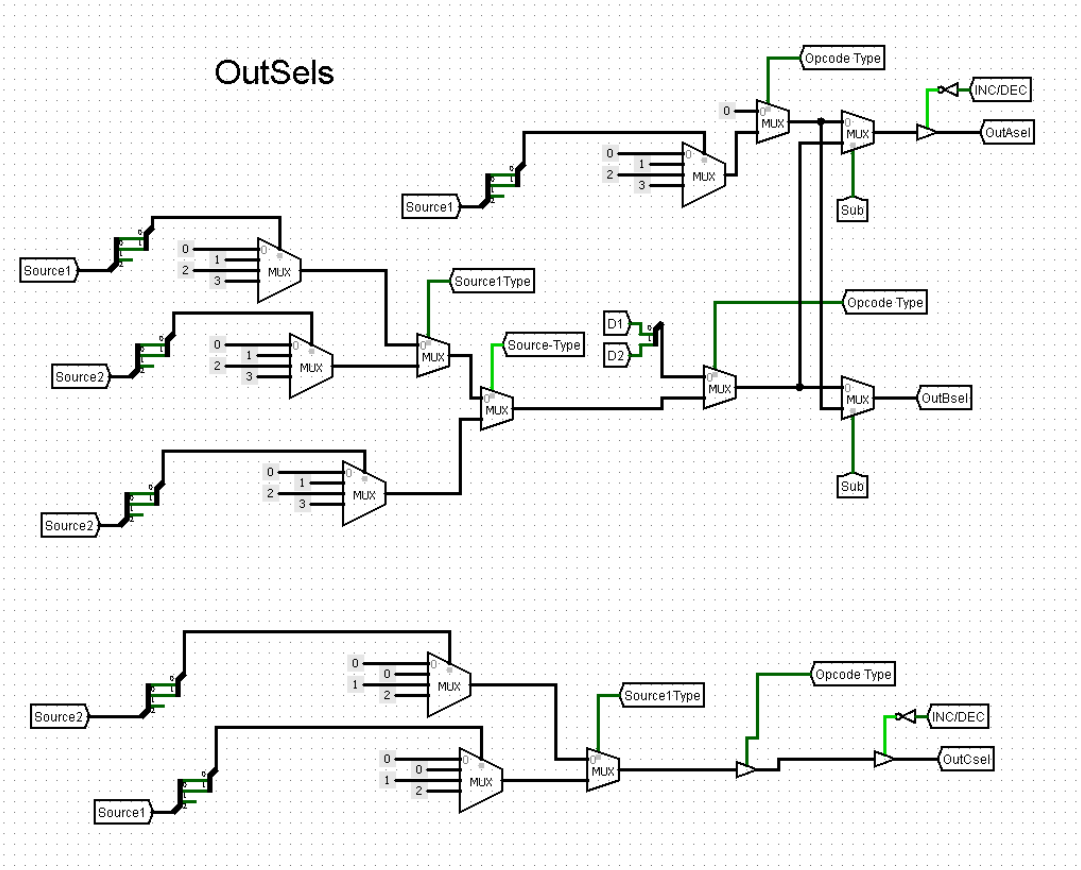

Figure 8: OutSels

In this part of the circuit; OutASel, OutBSel and OutCSel is decided according to operations and sources obtained from IR(0-7). OutASel mostly takes value with respect to Source1 value. If the operation is not an ALU operation then it becomes 00. Also if the operation is subtraction then we need to do (src2 - src1) to provide that source 2 value given from OutASel.

OutBSel is more complicated than OutASel because if one of the sources is address register then OutBSel has to give the other source as output. Intercrossing OutA and OutB are valid for OutBSel as mentioned above. Also, if Opcode Type is 0 which means instruction type is with addressing mode, it takes Rx values from IR (9-10) otherwise, it takes the value of appropriate source register. Choosing appropriate source register process is similar to the OutASel process only difference is it may take source 1 or source 2 value according to both of the registers are register file register or one of them is address register.

OutCSel is more simple than others. If Opcode Type is 0 then it does not take any value because no operation with address registers in those cases. Also, incrementing and

decrementing operations sets their OutCSels special for them in F operations so, the value does not take from here. If Source Type 1 is address register then OutCSel takes value according to Source 1 else it takes value according to Source 2.

2.4.5 Regsels and RAM Control

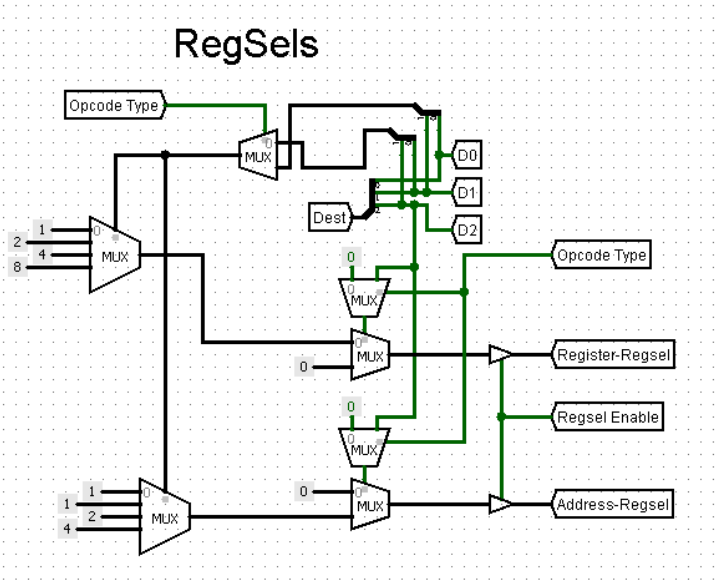

Figure 9: Regsels

Regsels are mostly used to load registers and increment or decrement the current value of registers. Therefore, Regsels take the value of destination register which will be loaded. A Regsel value depends only on destination registers. However, Regsels must take 0 value when they are not in use. Because it may lead to incorrect load operations. To supply that multiplexers are used concerning D2 which is 10^{th} bit of IR.

RAM Control takes value from micro-instruction. It sent from splitting of micro-instruction as 2 bits. Store and load take value according to its value.

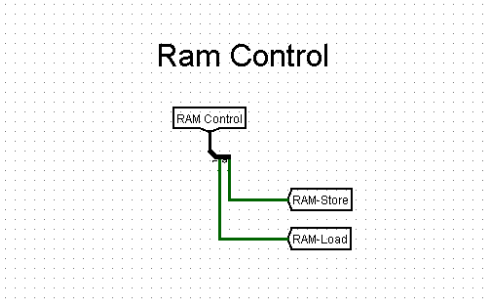

Figure 10: RAM Control

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 and in Table 2 calculates Fibonacci numbers and writes starting from 0xA0 to 0xA8.Then, the codes of the second part as can be seen in Table 3 and Table 4 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 (COPY)

Table 2: Finding the First 9 Fibonacci Numbers - The First Part Continue from 0x00

	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
LABEL4:	00	00	Load 0 to R0
	78	4a	IF Z=1;GO TO LABEL4
	00	00	This line is added to start with 0x50 line

Table 3: The second part from 0x50

	00	A0	Load A0 to R0
	16	00	Move R0 to AR
	04	08	Load 08 to R2
LABEL2:	03	00	Load M[AR] to R1
	18	20	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;

Table 4: The second part continue from 0x50

	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

Before the applying the fibonacci application, the memory is filled in Figure 11. Then, the result of the application is shown in Memory as shown in Figure 12.

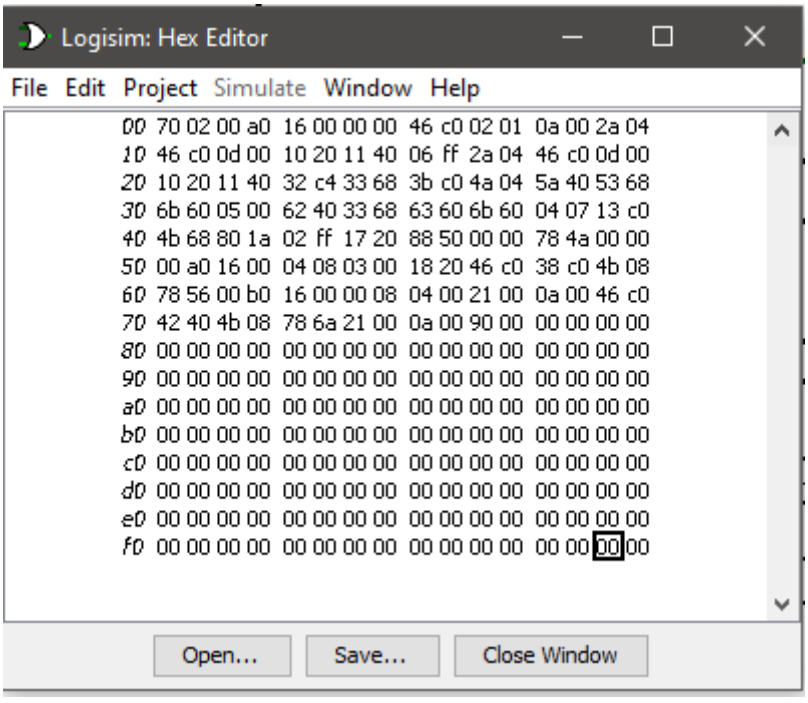

Figure 11: The Logisim Memory Code Before the Program

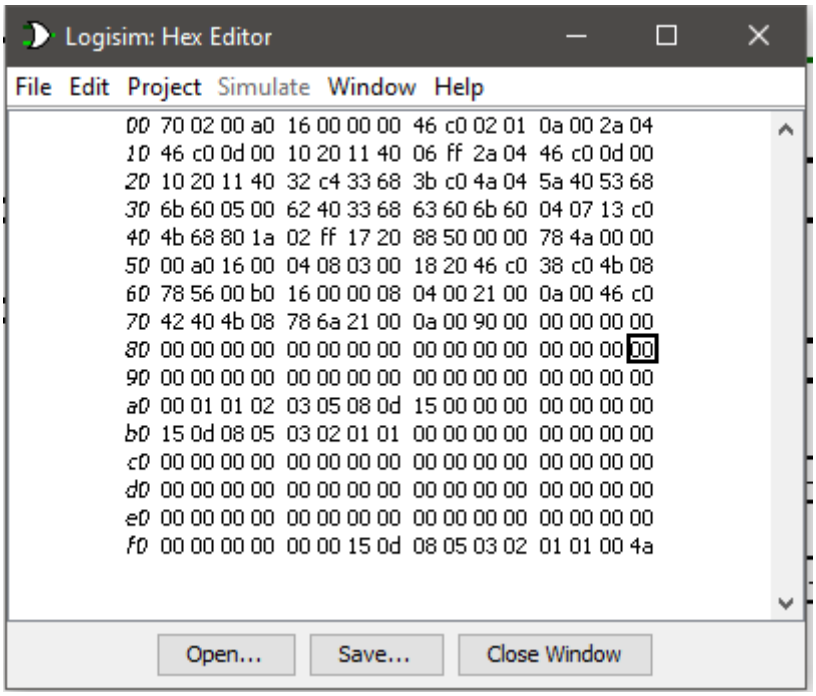

Figure 12: The Logisim Memory Code After the Program

4 DISCUSSION

At the start, we have apprehended which topics we have to cover for this homework. Principles of the software-based control units are written in the presentations but they only cover simple computer structures. For our case, our circuit is more complex than the example circuits. Then, we decided to come up with our design and we constructed our software-based system step by step from our sketches.

Firstly, we have started from micro-instruction format. In the presentation, there is a micro-instruction for each instruction case. To illustrate, there is a micro-instruction $R2 + R1 \rightarrow R3, AR + R1 \rightarrow R0$ etc. If we implement our circuit with this method, we have to write too many micro-instructions for each instruction. For this reason, we found a generalized micro-instruction format. This format has undergone many changes during the implementation.

After the construction of the main structure, we divided all opcodes and micro-operations to all group members and we gave a deadline for micro-operation coding. Finally, we have tested all codes with the test bench that we have written for HW3. Therefore, testing and finding a bug became easy. Also, the flexibility of the software-based control units made it easy to make changes.

In essence, the circuit that we put forward, have blended from a brainstorm that is

participated by all group members. We thought that it is very educational homework to understand micro-programs and it encouraged us to design an unprecedented circuit with our decisions.

5 CONCLUSION

In conclusion, we designed a micro-programmed CPU circuit that processes given 16-bit binary instructions in this project. We also compared a hard-wired circuit with a micro-programmed circuit and we realized both advantages and disadvantages of two circuit type. As an advantage, we could easily change the circuit when we got an error. As a disadvantage, we learned that software-based systems are more slower than hard-wired circuits. Another stuff we got from this project is that we had to push the limits of our creativity because of the lack of resources about software based systems. 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.