ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 2

DUE DATE : 09.06.2021

GROUP NO : G10

GROUP MEMBERS:

150170099 : BUSE AYYILDIZ

150170801 : ANIL KESKİN

150190015 : HIZIR EMİRHAN KESKİN

1 INTRODUCTION

In this second project, we are expected to design a hard-wired control unit for the circuit we have designed in the first project. In this project, instructions should be stored in memory in little endian order. Since the RAM we used in the first project has 8-bit output, we should load MSB and LSB separately in two clock cycles. In the first clock cycle, we will load LSB part of the instruction from an address A of the memory to LSB of IR. In the second clock cycle, we will load MSB part of the instruction from an address A+1 of the memory to MSB of IR. There are two types of instructions in this project: 1. Instructions with address reference 2. Instructions without address reference Instructions with address reference have the format below:

Figure 1: Instructions with address reference

Instructions without address reference have the format below:

Figure 2: Instructions without address reference

Also, there are 16 operations that we are expected to design. The table that shows the opcodes (in hex), symbols, addressing modes and descriptions for these operations is given below:

OPCODE (HEX)	SYMB	ADDRESSING MODE	DESCRIPTION
0x00	BRA	IM	PC ← Value
0x01	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x02	ST	D	Value ← Rx
0x03	MOV	N/A	DESTREG ← SRCREG1
0x04	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x05	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x06	NOT	N/A	DESTREG ← NOT SRCREG1
0x07	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x08	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x09	LSR	N/A	DESTREG ← LSL SRCREG1
0x0A	LSL	N/A	DESTREG ← LSR SRCREG1
0x0B	PUL	N/A	M[SP] ← Rx, SP ← SP - 1
0x0C	PSH	N/A	SP ← SP + 1, Rx ← M[SP]
0x0D	INC	N/A	DESTREG ← SRCREG1 + 1
0x0E	DEC	N/A	DESTREG ← SRCREG1 - 1
0x0F	BNE	IM	IF Z=0 THEN PC ← Value

Figure 3: Operation Table

Moreover, the table for selecting registers are given below:

REGSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

DESTREG/SRCREG1/SRCREG2	REGISTER
0000	PC
0001	PC
0010	AR
0011	SP
0100	R1
0101	R2
0110	R3
0111	R4

Figure 4: Table for Register Selection

And finally, the table that shows addressing modes and their values is given below:

ADDRESSING MODE	MODE	SYMB	Value
0	Direct	D	M[AR]
1	Immediate	IM	ADDRESS Field

Figure 5: Table for Addressing Modes

In the end, we are expected to control the inputs of the circuit we designed in the first project with outputs of the control unit that we will design in this second project.

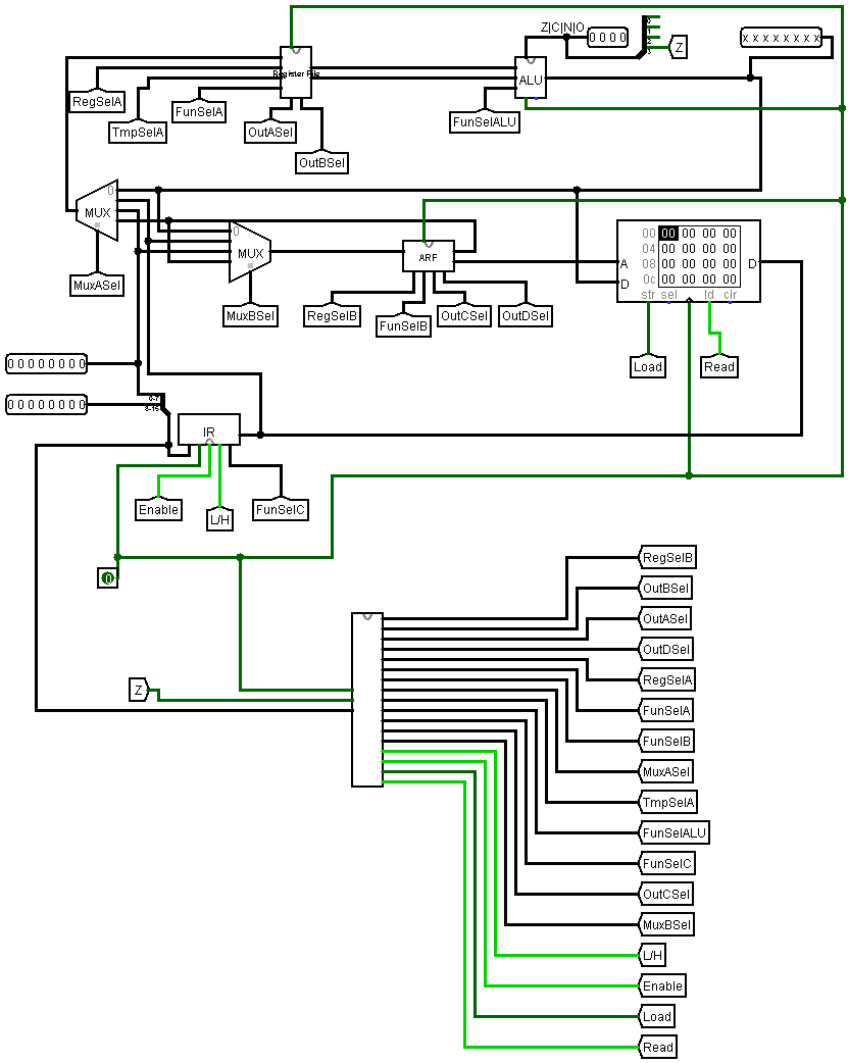

Figure 6: Last Circuit of the Project

2 PROJECTS PARTS

2.1 RegSel A

- This circuit's aim is to select the register to A. You can see the image of the circuit below.

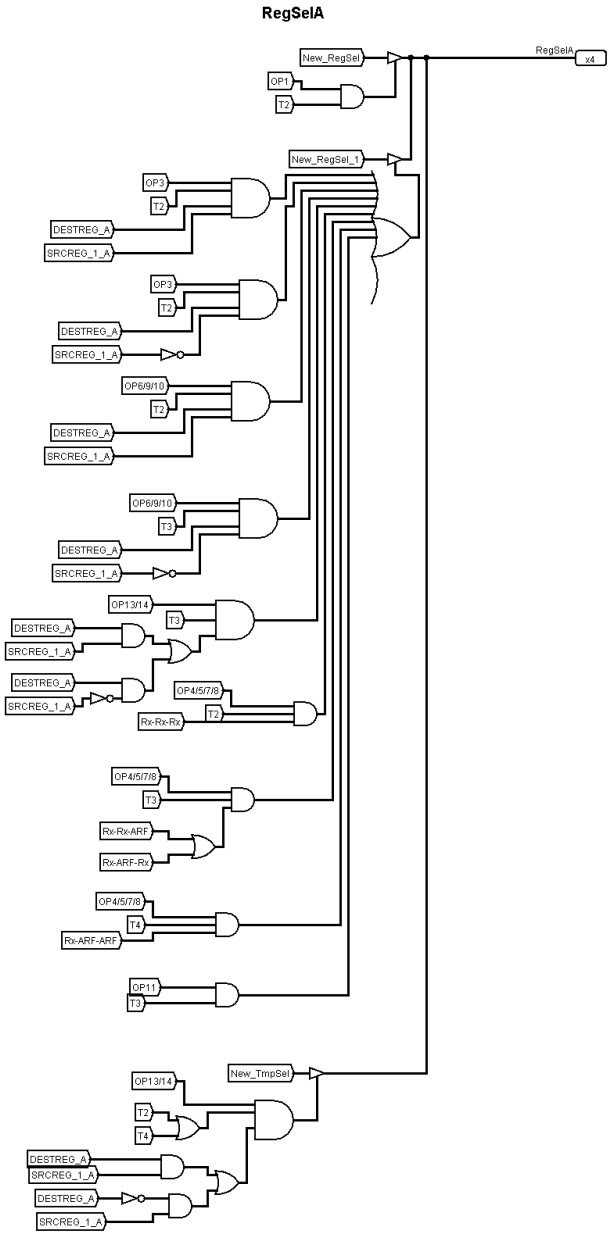

Figure 7: RegSel A

2.2 RegSel B

- This circuit's aim is to select the register to B. You can see the image of the circuit below.

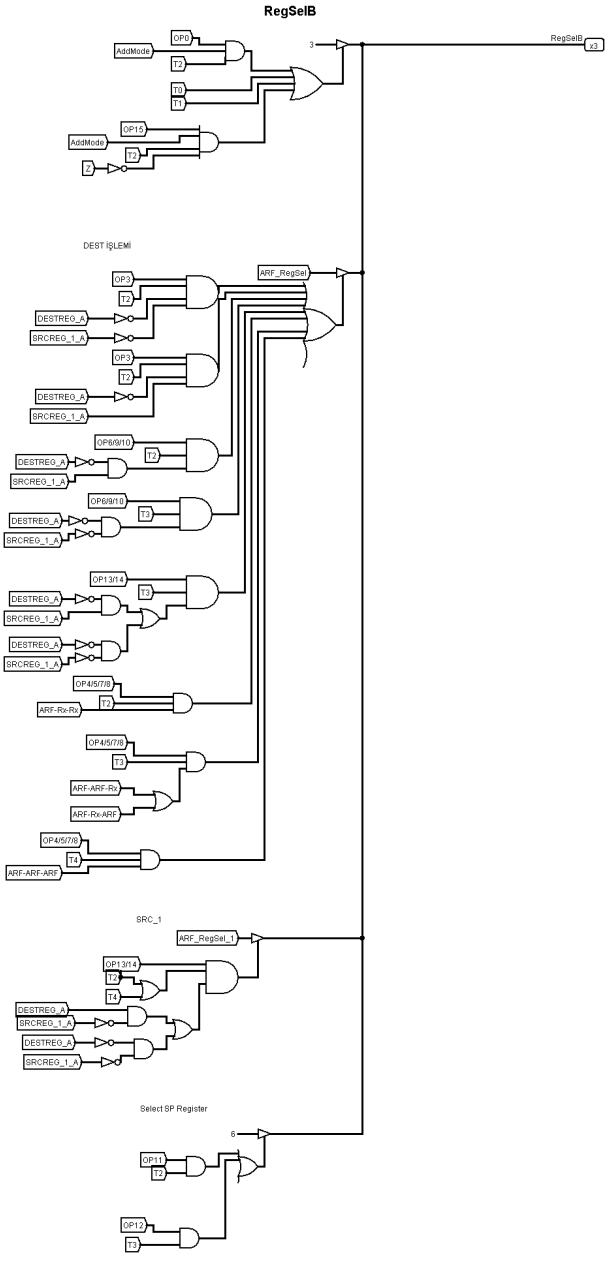

Figure 8: RegSel B

2.3 FunSel A

- This circuit's aim is to select the value to A. You can see the image of the circuit below.

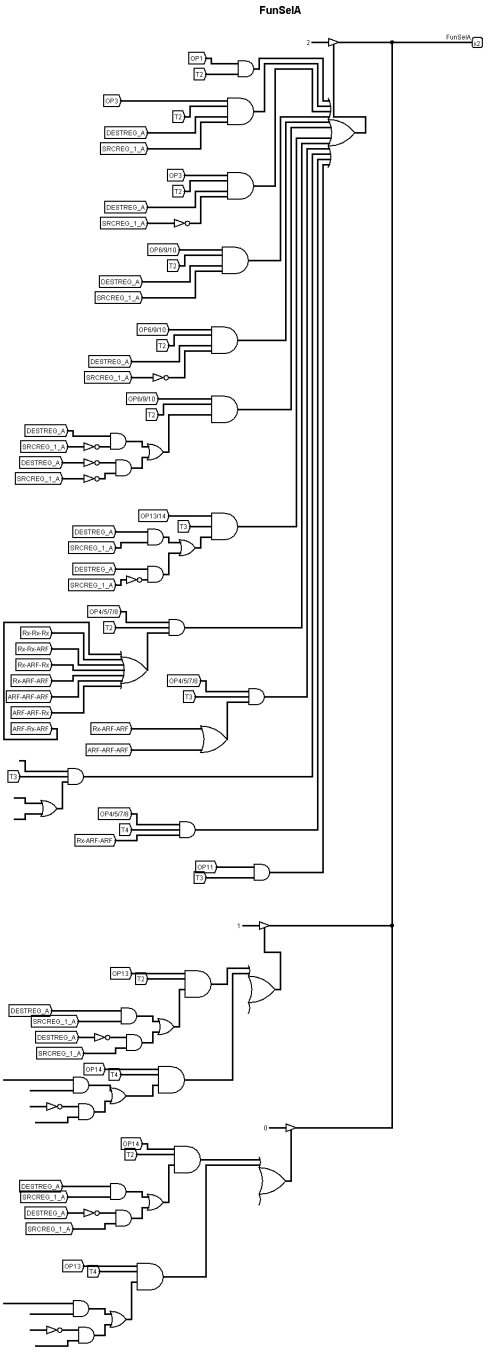

Figure 9: FunSel A

2.4 FunSel B

- This circuit's aim is to select the value to B. You can see the image of the circuit below.

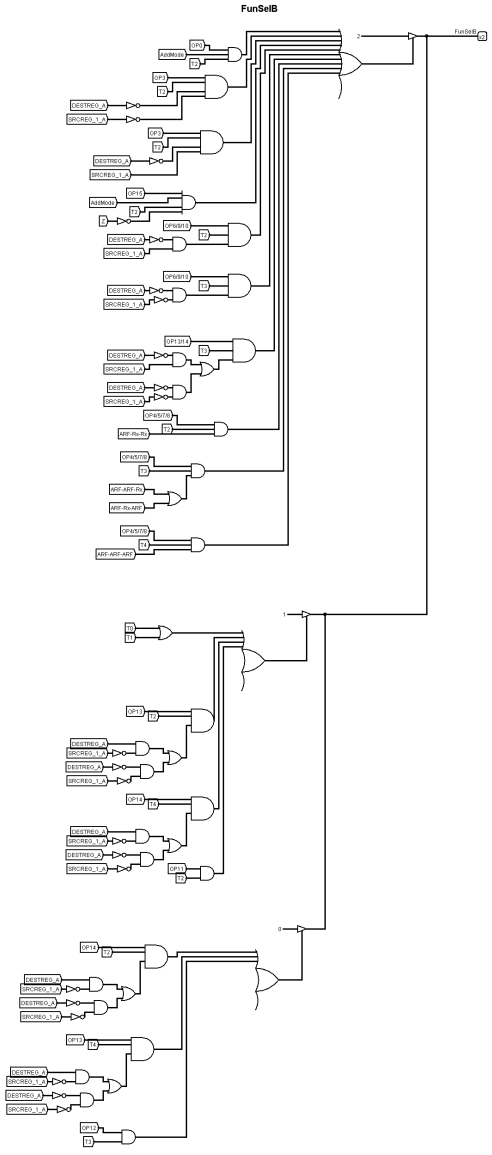

Figure 10: FunSel B

2.5 FunSel C

Select the value to C

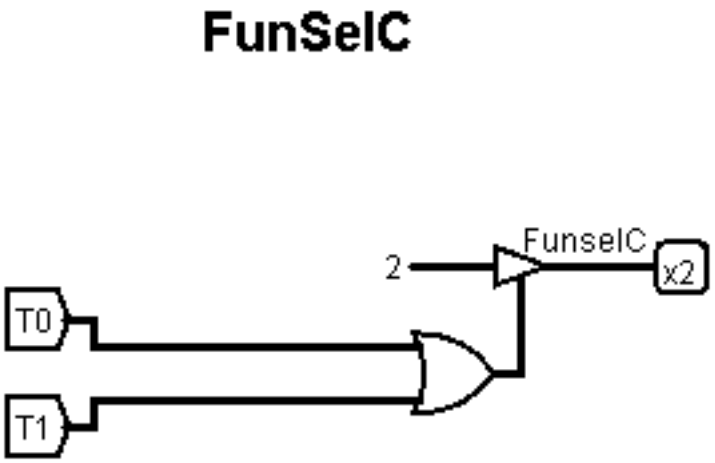

Figure 11: FunSel C

2.6 FunSel ALU

- This circuit's aim is to select the value to ALU. You can see the image of the circuit below.

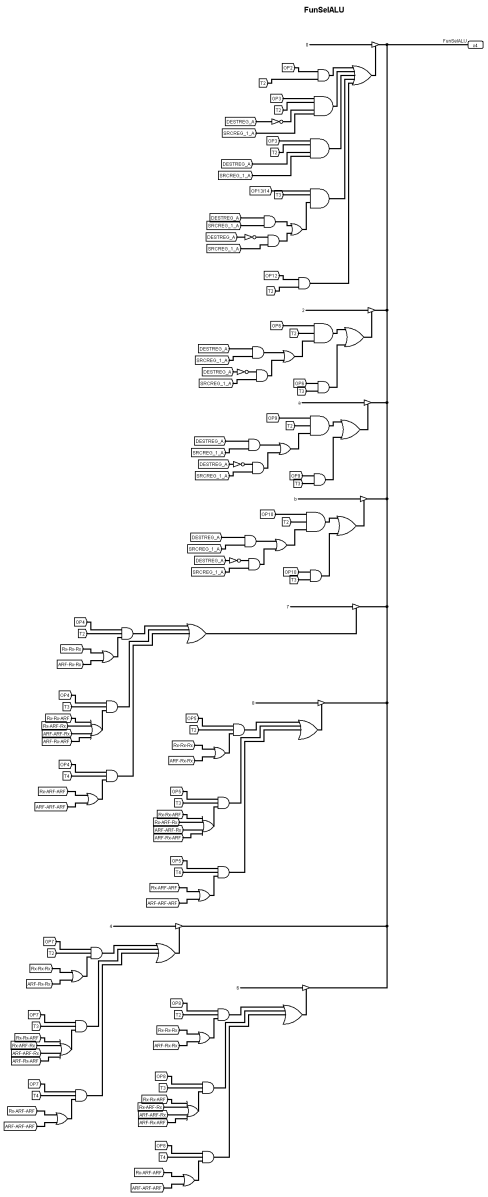

Figure 12: FunSel ALU

2.7 OutASel

- This circuit's aim is to select the output value to A. You can see the image of the circuit below.

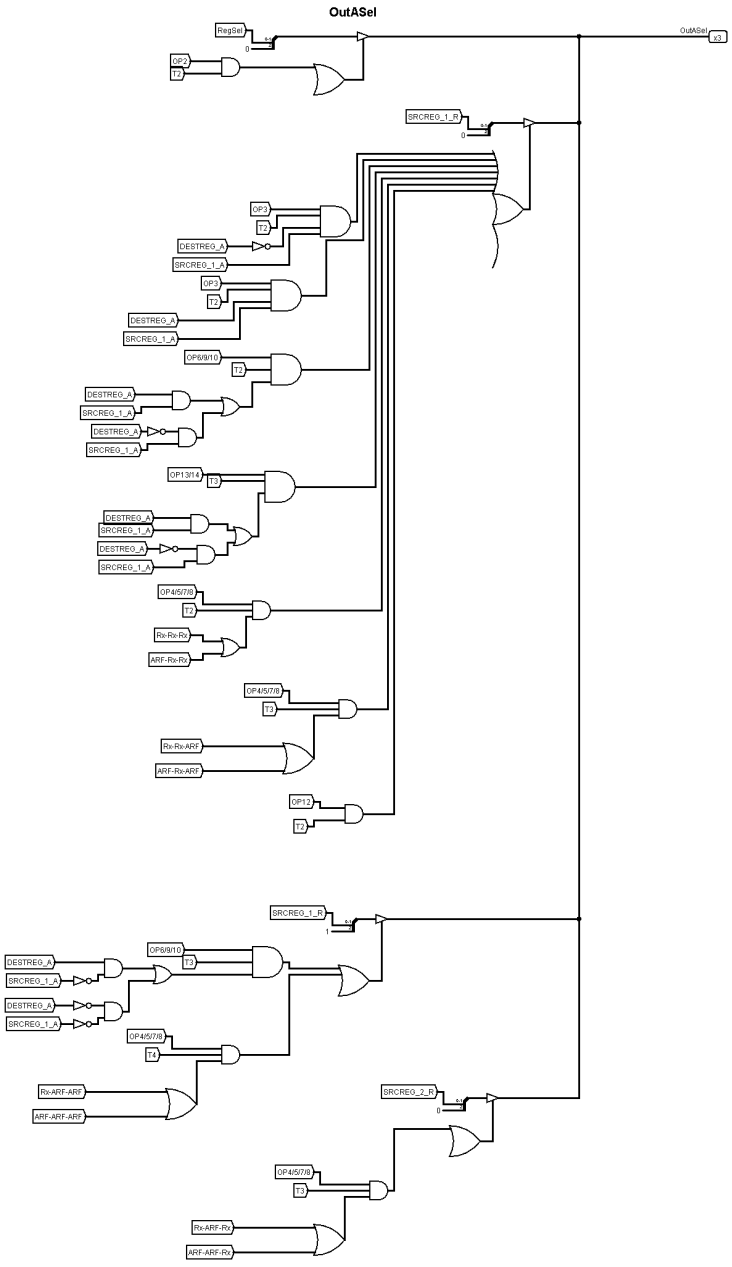

Figure 13: OutASel

2.8 OutBSel

- This circuit's aim is to select the output value to B. You can see the image of the circuit below.

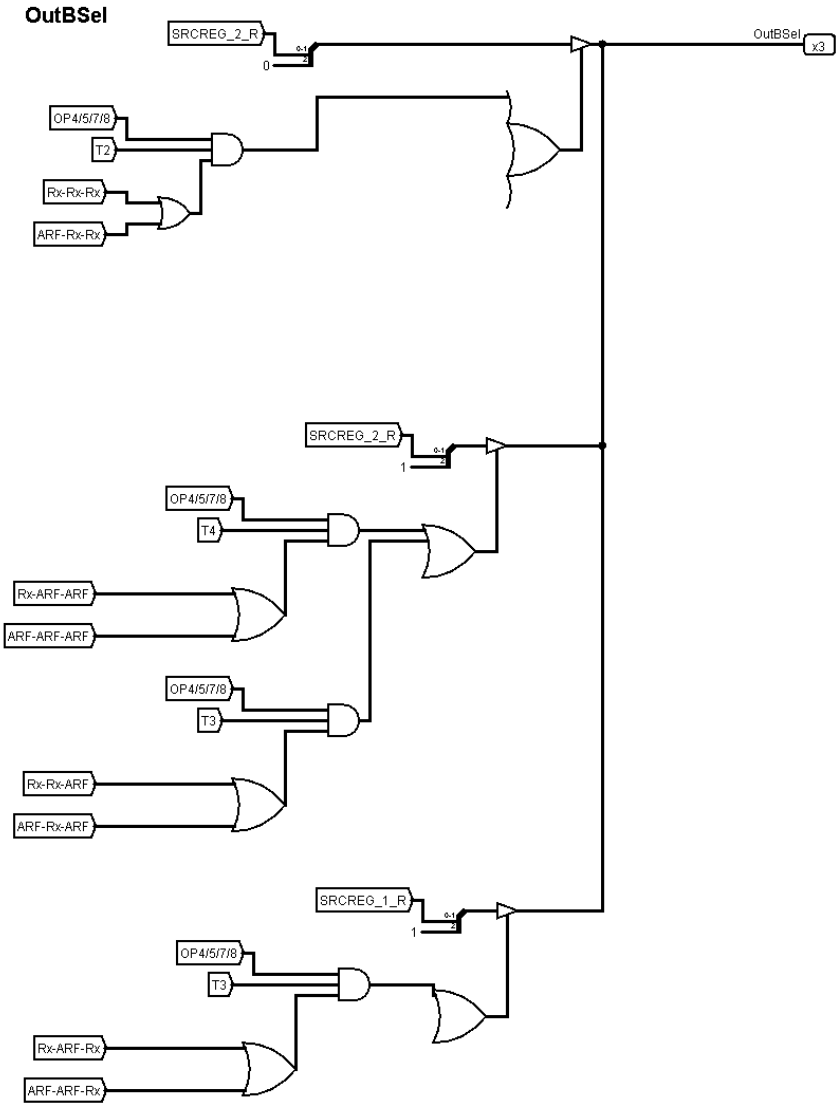

Figure 14: OutBSel

2.9 OutCSel

- This circuit's aim is to select the output value to C. You can see the image of the circuit below.

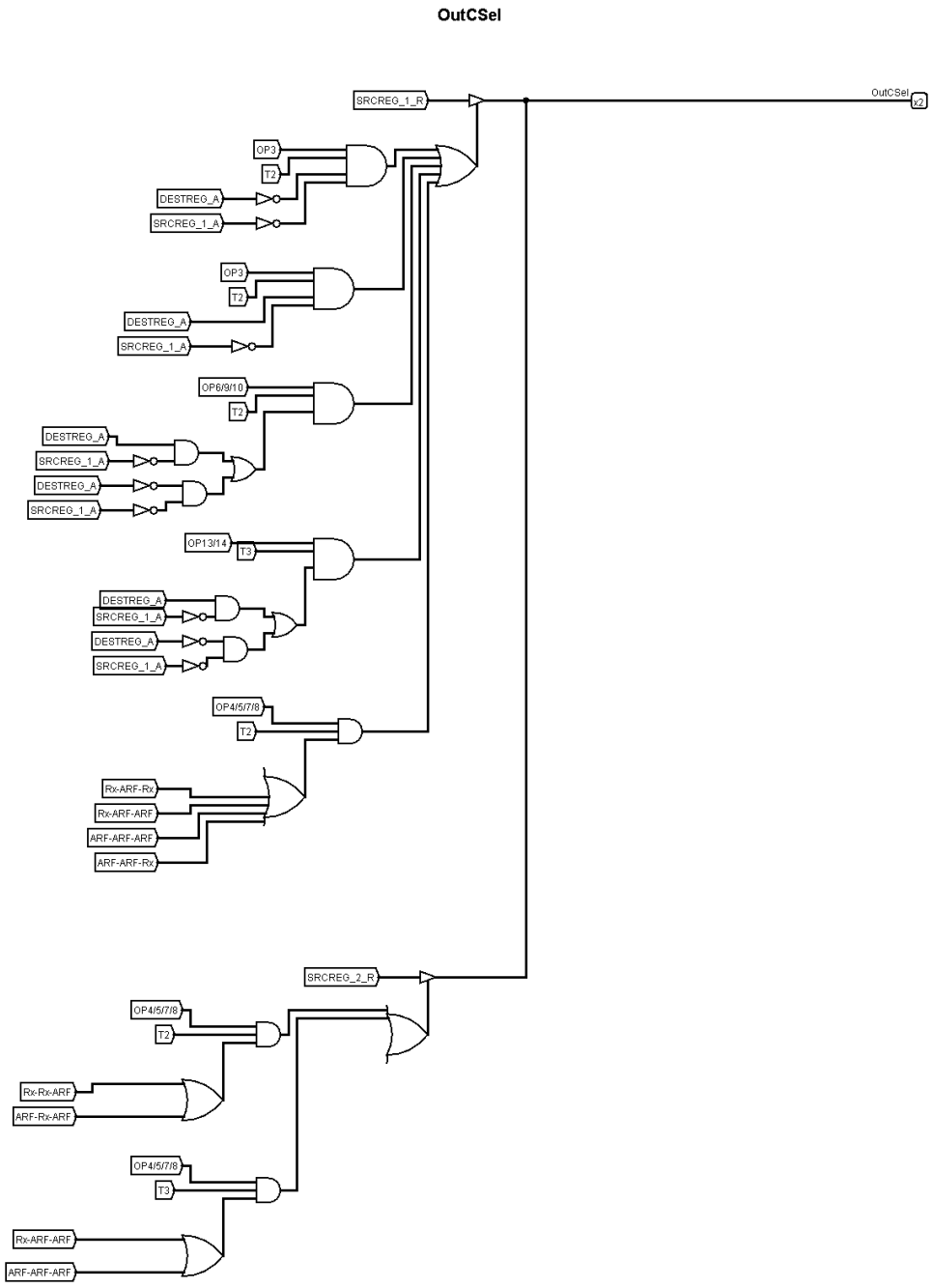

Figure 15: OutCSel

2.10 OutDSel

- This circuit's aim is to select the output value to D. You can see the image of the circuit below.

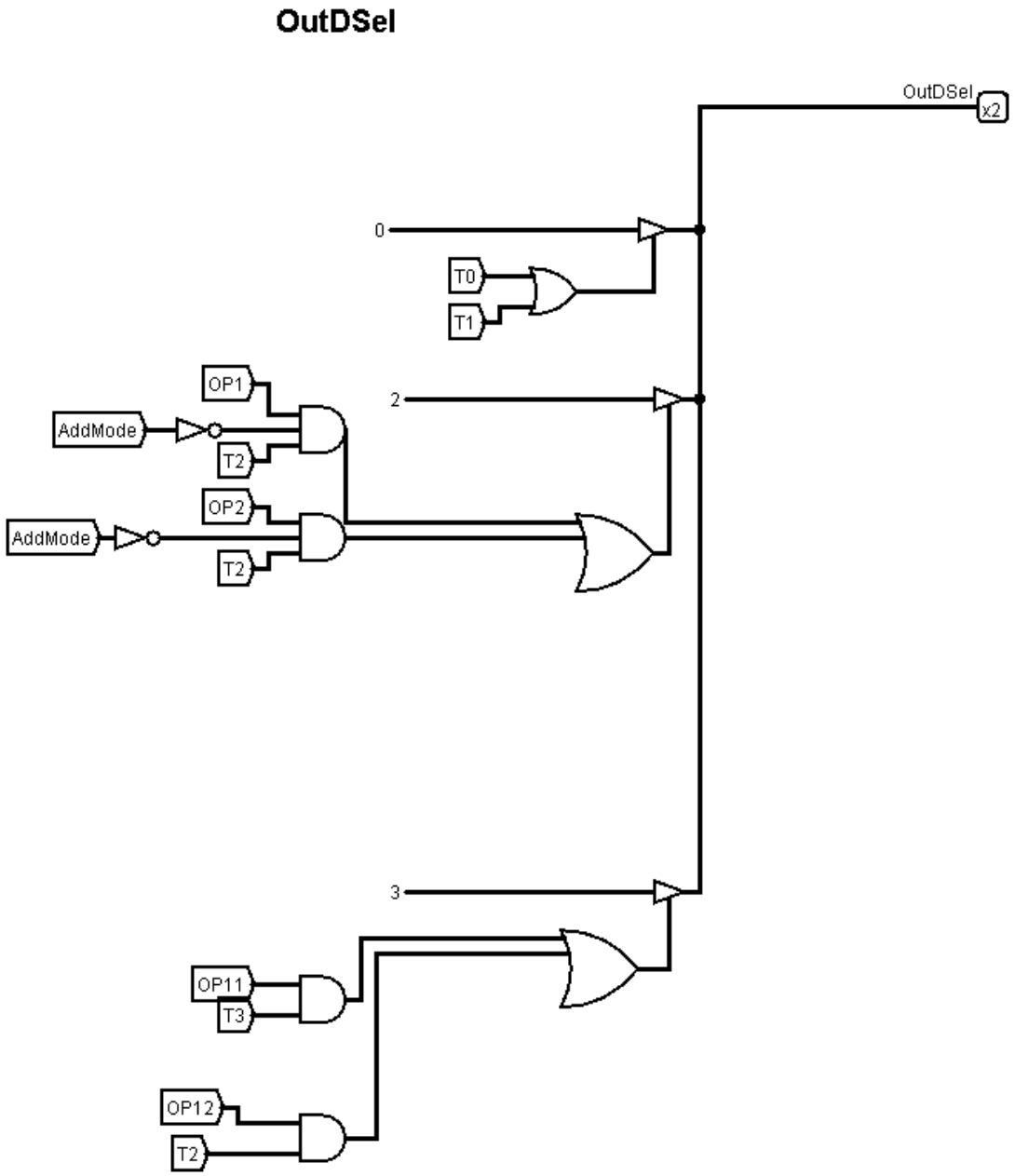

Figure 16: OutDSel

2.11 MuxASel

- This circuit's aim is to select the input to MuxA. You can see the image of the circuit below.

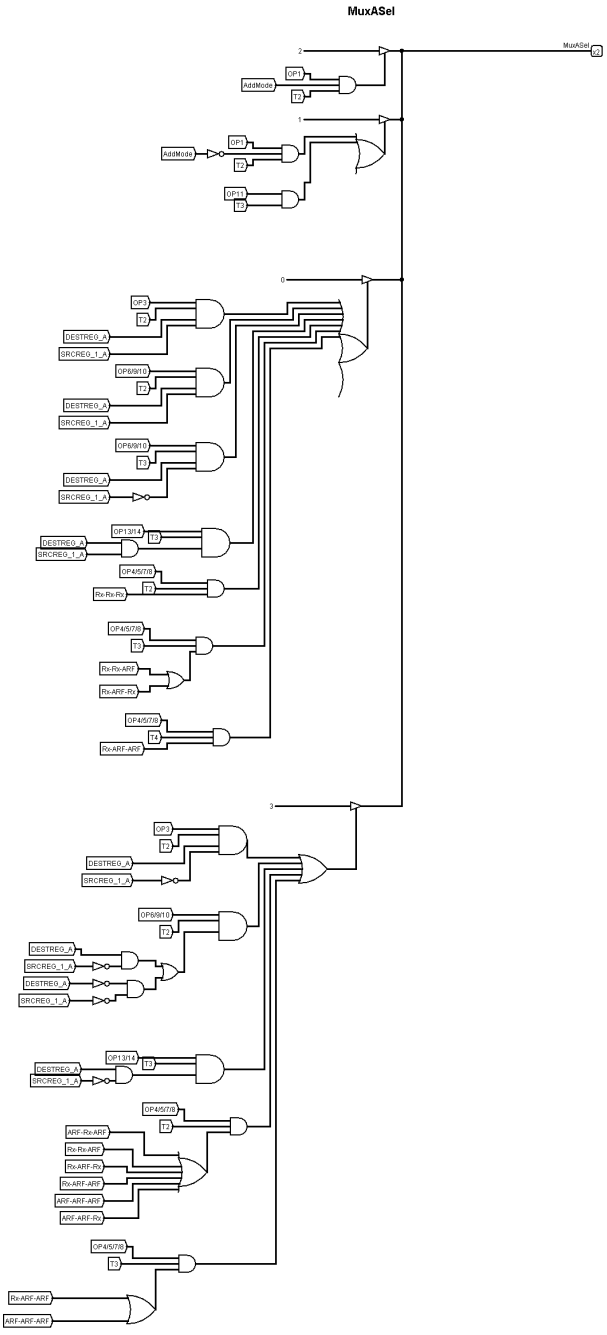

Figure 17: MuxASel

2.12 MuxBSel

- This circuit's aim is to select the input to MuxB. You can see the image of the circuit below.

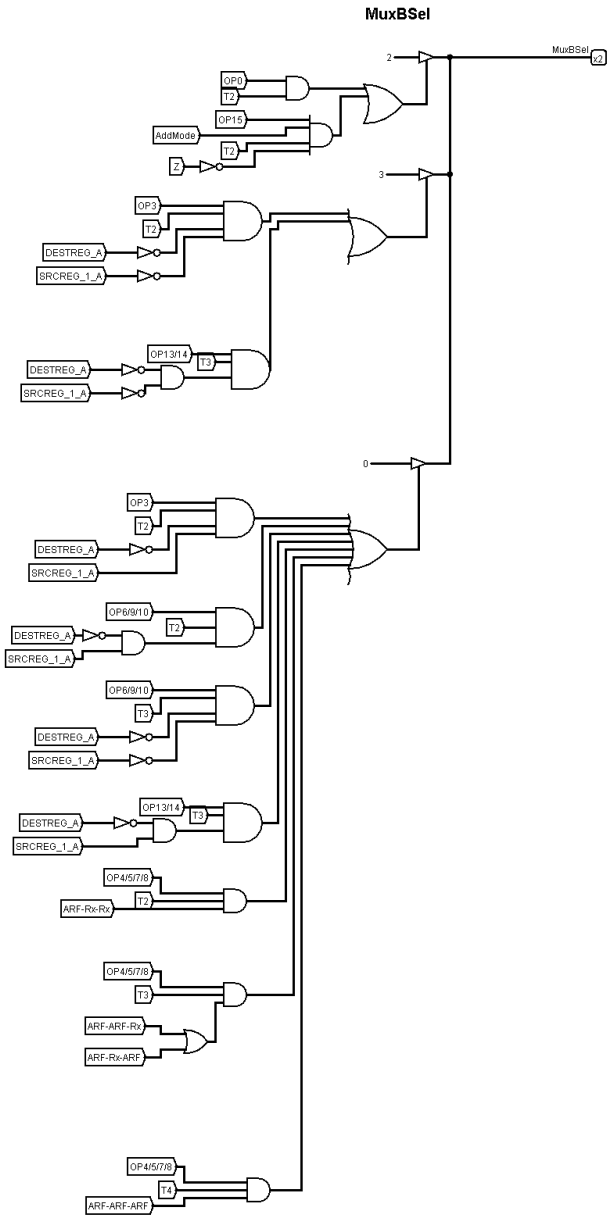

Figure 18: MuxBSel

2.13 TmpSelA

- This circuit's aim is to store the data temporarily. You can see the image of the circuit below.

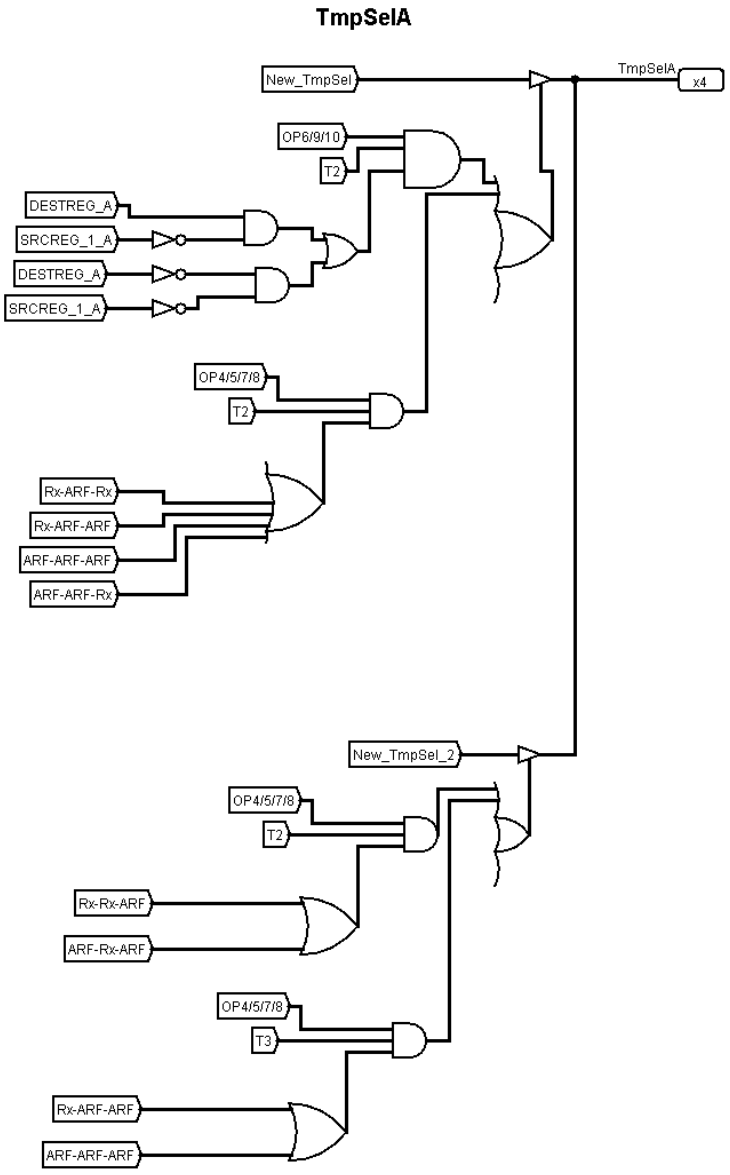

Figure 19: TmpSelA

2.14 Load

- Load Operation

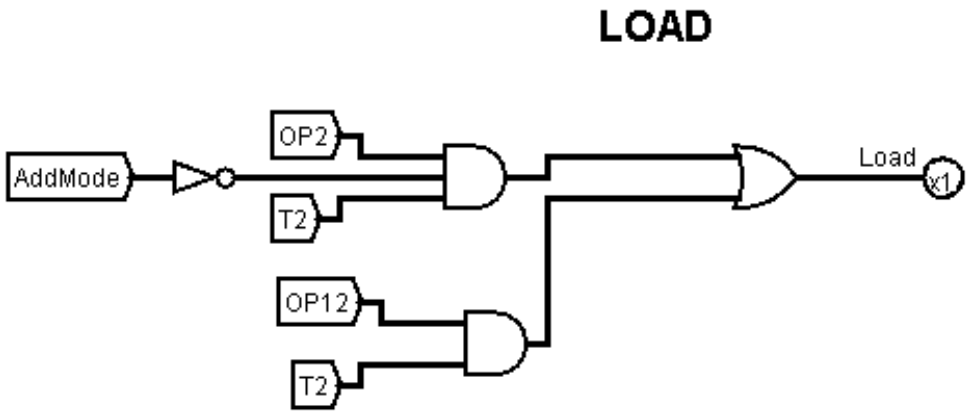

Figure 20: Load

2.15 Read

- Read Operation

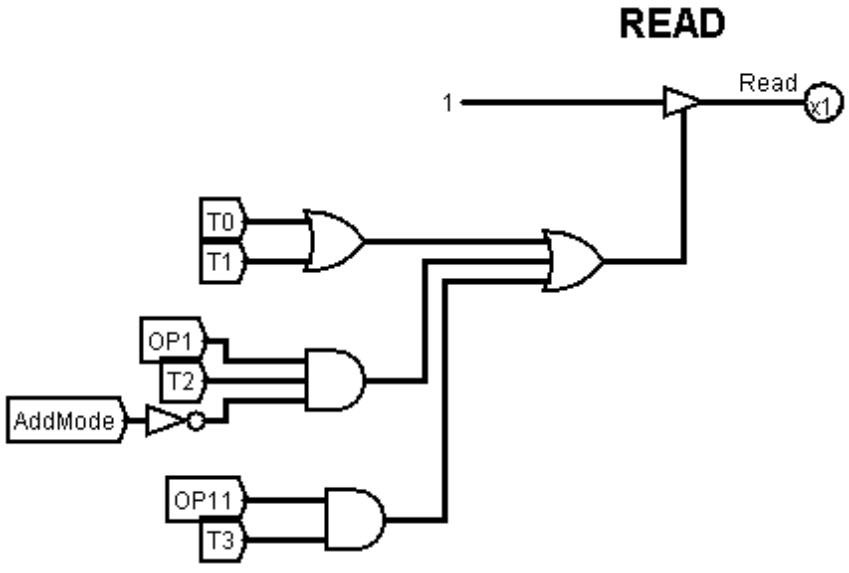

Figure 21: Read

2.16 Others Operations

2.16.1 CLOCK CYCLES

- This circuit's aim is to determine the "T" value, such as T0, T1, according to Clock value. Also, "Finish" determines the operation clock number according to OpCode. You can see the image of the circuit below.

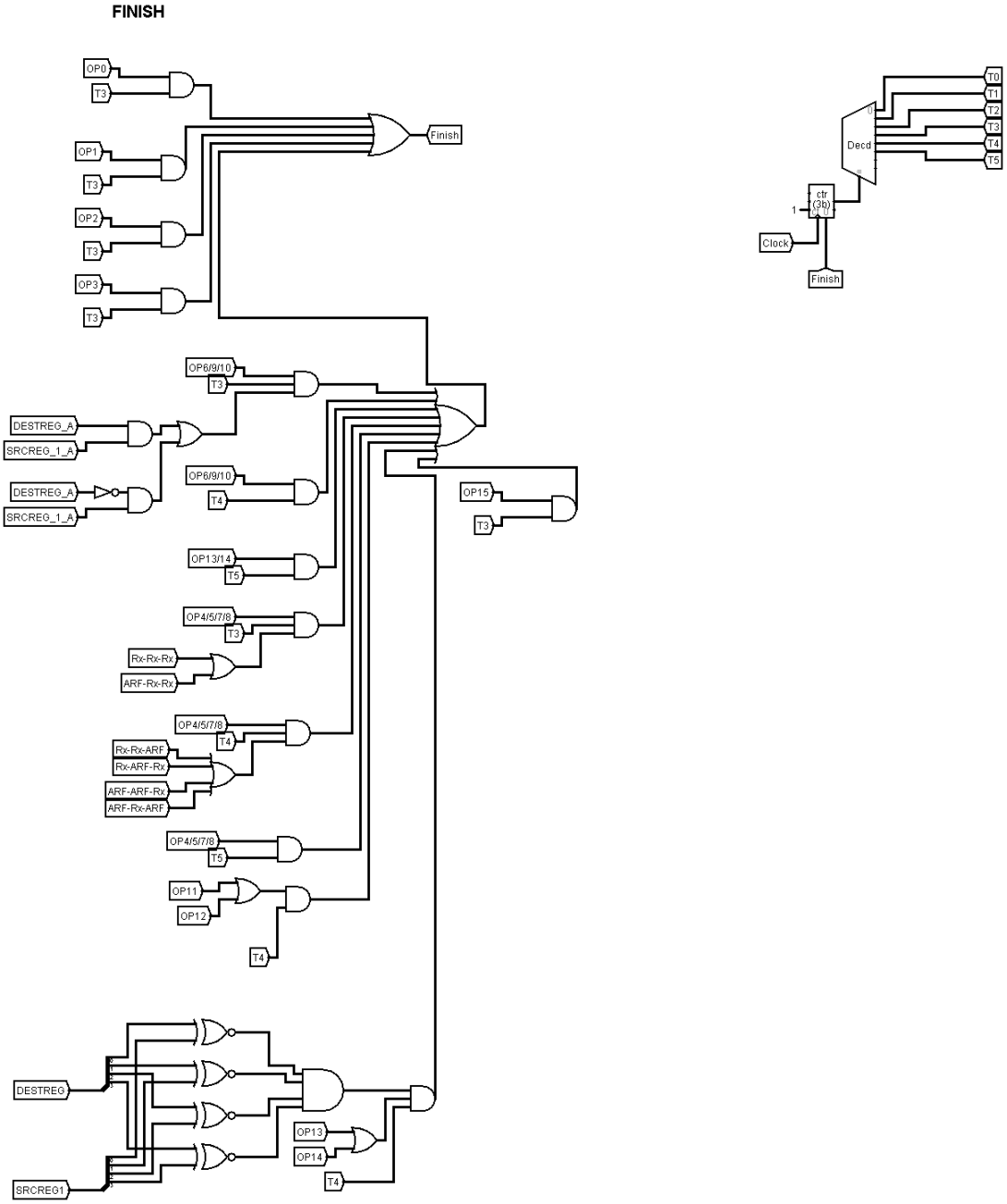

Figure 22: Clock Cycles

2.16.2 Determining OPCODE

- This circuit's aim at left side is to determine the OpCode according to IR(15-12) and at right side is to determine RegSel, AddressMode, Address, SRCREG1, SRCREG2, DESTREG according to IR(11-0). In Opcode 8, the value of SRC1 must be SRC2. At the same time, the value of SRC2 must also be SRC1. You can see the image of the circuit below.

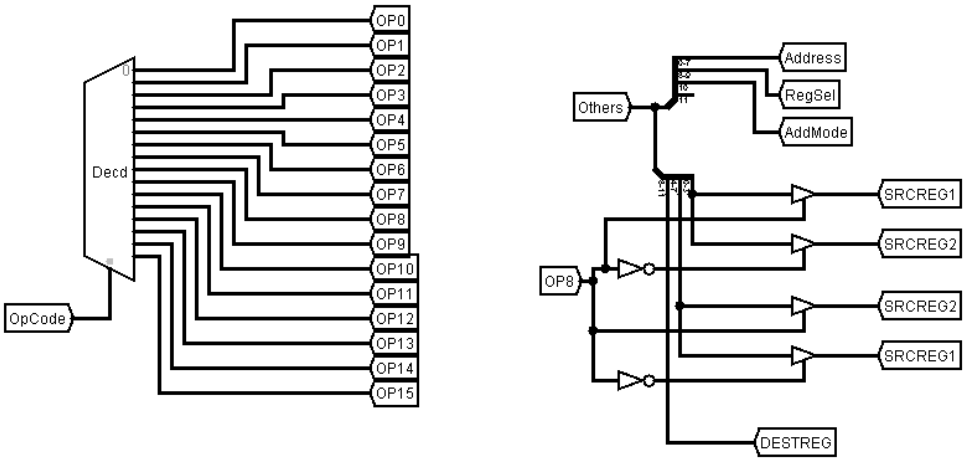

Figure 23: OpCode

2.16.3 Determining DEST SRC1 SRC2

- Split Operations

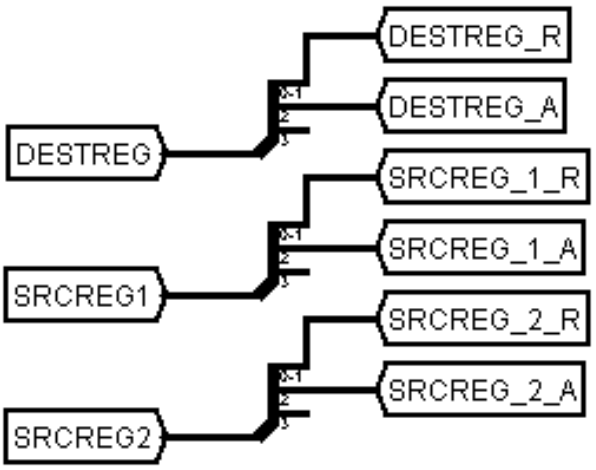

Figure 24: Splitter for Determine

- Multiplexers

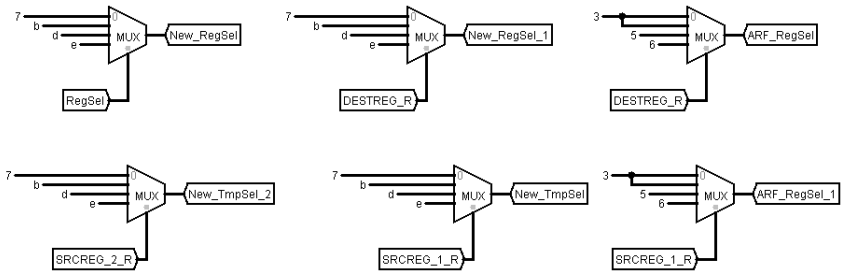

Figure 25: MUX for Determine

- Determining DEST SRC1 SRC2 with Using AND Gate

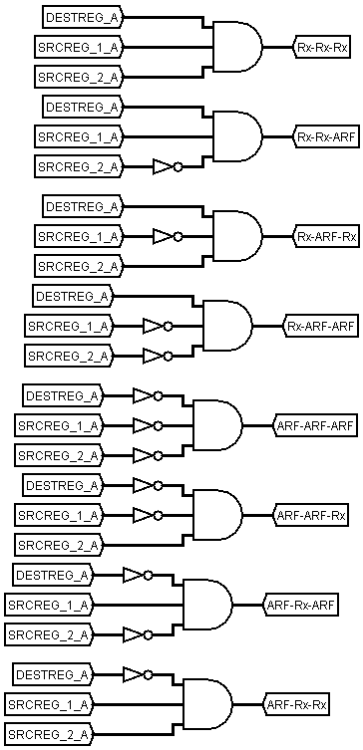

Figure 26: AND Gates

3 RESULTS

3.1 T0-T1

The T0 and T1 operations are the first-two CLOCK values.

T0 : Load IR(7-0)
outDSel = 00(PC)
Read = 1
L/H = 1
Enable = 1
CLOCK
T1 : Load IR(15-8)
outDSel = 00(PC)
Read = 1
L/H = 0
Enable = 1
CLOCK

3.2 0x00 – BRA IM

PC <- Address Field
MuxBSel = 10
RegSelB = 001(PC)
FunSelB = 10(Load)
CLOCK

3.3 0x01 – LD - IM, D

There are 2 operation for this Opcode (IM and D).

IM(Address)	—	D(M[AR])
Rx <- Address	—	**Rx <- M[AR]**
MuxASel=10	—	outDSel = 10(AR)
RegSelA = Rx	—	Read = 1
FunSelA = 10	—	MuxASel = 01
CLOCK	—	RegSelA=Rx
		FunSelA = 10
		CLOCK

3.4 0x02 – ST – D

M[AR] <- Rx
outASel = Rx
FunSelALU = 0000
outDSel = 10(AR)
Load = 1
CLOCK

3.5 0x03 – MOV, 0x06 – NOT, 0x09 – LSL, 0x0A – LSR

These operations were done together with all possibilities in mind. This table was made for Opcode 6 with all possibilities in mind. For other operations, only the value of FunSelALU varies. For example, for Opcode 10, the FunSelALU value should be 1011.

Rx <- NOT Rx	**Rx <- NOT ARF**	**ARF <- NOT ARF**	**ARF <- NOT Rx**
outASel = SRCREG1$_R$	outCSel = SRCREG$_R$	outCSel = SRCREG$_R$	outASel = SRCREG1$_R$
FunSelALU = 0010	MuxASel = 11	MuxASel = 11	FunSelALU = 0010
MuxASel = 00	TmpSelA = SRC	TmpSelA = SRC	MuxBSel = 00
RegSelA = DEST	FunSelA =10	FunSelA =10	RegSelB = DEST
FunSelA = 10	**CLOCK**	**CLOCK**	FunSelB = 10
CLOCK	outASel = TEMP	outASel = TEMP	**CLOCK**
	FunSelALU = 0010	FunSelALU = 0010	
	MuxASel = 00	MuxBSel = 00	
	RegSelA = DEST	RegSelB = DEST	
	FunSelA = 10	FunSelB = 10	
	CLOCK	**CLOCK**	

3.6 0x04 – AND, 0x05 – OR, 0x07 – ADD, 0x08 – SUB

These operations were done together with all possibilities in mind. This table was made for Opcode 4 with all possibilities in mind. For other operations, only the value of FunSelALU varies. For example, for Opcode 5, the FunSelALU value should be 1000. In Opcode 8, the value of SRC1 must be SRC2. At the same time, the value of SRC2 must also be SRC1. We assigned this automatically when performing Opcode operations.

Rx<- Rx & Rx	Rx <- Rx & ARF	Rx<- Arf &Rx	Rx<-ARF & ARF
outASel = SRC1	outCSel = SRC2	outCSel=SRC1	outCSel=SRC1
outBSel = SRC2	muxASel =11	muxASel = 11	muxASel=11
FunSelALU= 0111	TmpSelA=SRC2	TmpSelA=SRC1	TmpSelA=SRC1
MuxASel =00	FunSelA = 10	FunSelA =10	FunSelA=10
RegSelA = DEST	**CLOCK**	**CLOCK**	**CLOCK**
FunSelA = 10	outASel =SRC1	outAsel =SRC2	outCSel=SRC2
CLOCK	outBSel = SRC2 (Temp)	outBSel = SRC1 (Temp)	muxASel=11
	FunSelALU=0111	FunSelALU 0111	TmpSelA=SRC2
	MuxASel = 00	MuxASel 00	FunSelA=10
	RegSelA= DEST	RegSelA DEST	**CLOCK**
	FunSelA = 10	FunSelA 10	outASel = SRC1 (Temp)
	CLOCK	**CLOCK**	outBSel = SRC2 (Temp)
			FunSelALU 0111
			MuxASel 00
			RegSelA DEST
			FunSelA 10
			CLOCK

Figure 27: All Cases 1 for OPCODE-04-05-07-08

ARF<- ARF &ARF	ARF<- ARF &Rx	ARF<-Rx&ARF	ARF <-Rx & Rx
outCSel SRC1	outCSel SRC1	outCSel SRC2	outASel SRC1
muxAsel 11	MuxASel 11	MuxASel 11	outBSel SRC2
TmpSelA SRC1	TmpSelA SRC1	TmpSelA SRC2	FunSelALU 0111
FunSelA 10	FunSelA 10	FunSelA 10	MuxBsel 00
CLOCK	**CLOCK**	**CLOCK**	RegSelB DEST
outCSel SRC2	outASel SRC2	outASel SRC1	FunBSel 10
muxAsel 11	outBSel SRC1 (Temp)	outBSelSRC2 (Temp)	**CLOCK**
TempSelA SRC2	FunSelALU 0111	FunSelALU 0111	
FunSelA 10	muxBSel 00	muxBSel 00	
CLOCK	RegSelB DEST	RegSelB DEST	
outAsel = SRC1 (Temp)	FunSelB 10	FunSelB 10	
outBSel=SRC2 (Temp)	**CLOCK**	**CLOCK**	
FunSelALU 0111			
MuxBSel 00			
RegBSel DEST			
FunBSel 10			
CLOCK			

Figure 28: All Cases 2 for OPCODE-04-05-07-08

3.7 0x0B PUL

SP <- SP + 1
RegSelB=110 (SP)
FunBSel=01 (INC)
CLOCK
Rx <-M[SP]
outDSel = 11 (SP)
Read = 1
MuxASel = 01
RegSelA = DESTREG
FunSelA = 10
CLOCK

3.8 0x0C PSH

M[SP] <-Rx
outDSel = 11 (SP)
outASel = SRCREG1
FunSelALU = 0000
Load = 1
CLOCK
SP <- SP - 1
RegSelB=110 (SP)
FunBSel=00 (DEC)
CLOCK

3.9 0x0D INC, 0x0E DEC

Rx <- Rx +1	Rx <- ARF +1	ARF <- ARF +1	ARF <- Rx +1
RegSelA = SRCREG_R	RegSelB = SRCREG_R	RegSelB = SRCREG_R	RegSelA = SRCREG_R
FunSelA =01	FunSelB = 01	FunSelB = 01	FunSelA =01
CLOCK	**CLOCK**	**CLOCK**	**CLOCK**
outASel =SRC ye göre	outCSel = SRC ye göre	outCSel = SRC ye göre	outASel =SRC ye göre
FunSelALU = 0000			FunSelALU = 0000
MuxASel = 00	MuxAsel =11	MuxBSel = 11	MuxBSel = 00
RegSelA = DEST	RegSelA = DEST	RegSelB = DEST	RegSelB = DEST
FunSelA = 10	FunSelA = 10	FunSelB = 10	FunSelB = 10
CLOCK	**CLOCK**	**CLOCK**	**CLOCK**
RegSelA = SRCREG_R	RegSelB= SRCREG_R	RegSelB= SRCREG_R	RegSelA = SRCREG_R
FunSelA = 00	FunSelB = 00	FunSelB = 00	FunSelA = 00
CLOCK	**CLOCK**	**CLOCK**	**CLOCK**

Figure 29: All Cases for OPCODE 0x0D and 0x0E

3.10 0x0F – BNE – IM

We performed the same operations as Opcode 0 (BRA) but Z value must be 0.

PC <- Address Field
MuxBSel = 10
RegSelB = 001(PC)
FunSelB = 10 (Load)
CLOCK

3.11 TEST

We have tested the given example in the end of the project. Initial values of RAM is given below:

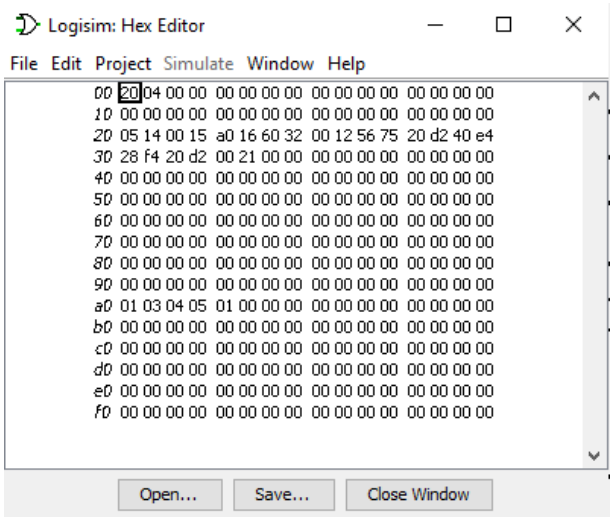

Figure 30: Initial values of RAM before clocks

After given enough amount of clocks, contents of RAM is given below:

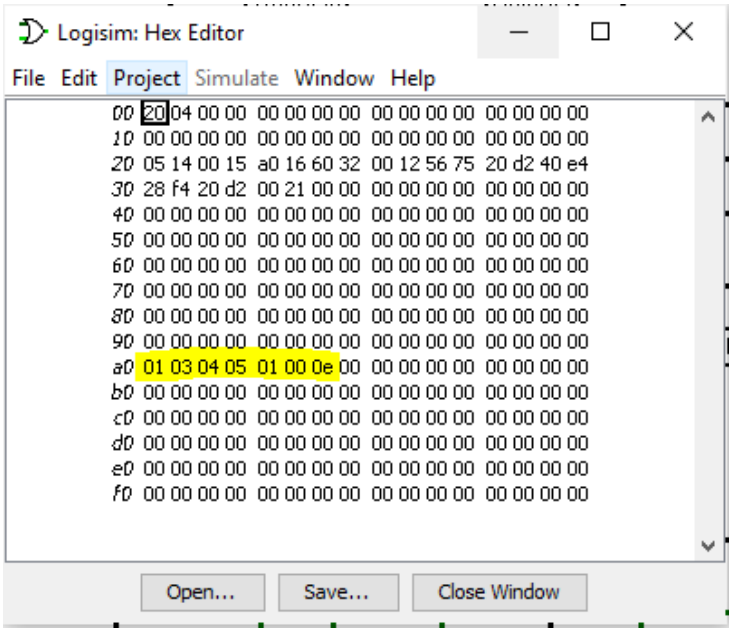

Figure 31: Contents of RAM after clocks

4 DISCUSSION

In 0x00 operation, our goal is loading value to the PC (PC <- Value). Since the addressing mode is IM, our value is address field and we should load this address field

to the PC. Therefore we set our MuxBSel, RegSelB and FunSelB as mentioned in results part. This part was quite easy and we finished this part successfully.

In 0x01 operation, our goal is loading value to the Rx (Rx <- Value). Since the addressing mode can be both IM or D, we should think about two possibilities. If addressing mode is IM, then we should load address field. Therefore, we set our MuxASel, RegSelA and FunSelA as mentioned in results part. But if addressing mode is D, then we should load M[AR]. Therefore, we set our OutDSel, Read, MuxASel, RegSelA and FunSelA as mentioned in results part. Thinking about two possibilities was a bit challenging but we implemented this part successfully.

In 0x02 operation, our goal is loading Rx to the value (Value <- Rx). With this operation we are storing the value of Rx in the memory. Since the addressing mode is D, our value is M[AR] and we should load M[AR] to the Rx. Therefore, we set our OutASel, FunSelALU, OutDSel and Load as mentioned in results part. This operation was also easy and we finished this part successfully.

In 0x03, 0x06, 0x09, 0x0A operations, even though our goal is different (For 0x03: DESTREG <- SRCREG1 ; For 0x06: DESTREG <- NOT SRCREG1 ; For 0x09: DESTREG <- LSL SRCREG1 ; For 0x0A: DESTREG <- LSR SRCREG1), the implementation methods are quite similar. The only difference of these implementations is setting the input of FunSelALU. Therefore, we implemented these operations together. However, there is a challenging issue in this part. Since our DESTREG and SRCREG1 can be choosen from both Register File and Address Register File, there are four possibilities in this operations and we considered all of these possibilities. If both DESTREG and SRCREG1 are choosen from Register File; we set OutASel, FunSelALU, MuxASel, RegSelA and FunSelA. If DESTREG is choosen from Register File and SRCREG1 is choosen from Address Register File; we set OutCSel, MuxASel, TmpSelA and FunSelA, then give a clock, then we set OutASel, FunSelALU, MuxASel, RegSelA and FunSelA. If both DESTREG and SRCREG1 are choosen from Address Register File; we set OutCSel, MuxASel, TmpSelA and FunSelA, then give a clock, then we set OutASel, FunSelALU, MuxBSel, RegSelB and FunSelB. If DESTREG is choosen from Address Register File and SRCREG1 is choosen from Register File; we set OutASel, FunSelALU, MuxBSel, RegSelB and FunSelB. Calculating and implementing these four possibilities separately was quite hard but we implemented these four operations successfully with extra effort.

In 0x04, 0x05, 0x07, 0x08 operations, even though our goal is different (For 0x04: DESTREG <- SRCREG1 AND SRCREG2 ; For 0x05: DESTREG <- SRCREG1 OR SRCREG2 ; For 0x07: DESTREG <- SRCREG1 + SRCREG2 ; For 0x08: DESTREG <- SRCREG2 - SRCREG1), the implementation methods are quite similar. The only difference of these implementations is setting the input of FunSelALU. Therefore, we

implemented these operations together. However, there is a very challenging issue in this part. Since our DESTREG, SRCREG1 and SRCREG2 can be choosen from both Register File and Address Register File, there are eight possibilities in this operations and we considered all of these possibilities. As we explained in previous paragraph, we used the same logic and we did our implementations separately for each possibility as mentioned in results part. However, we realized that there is an extra problem in this part. In operation 0x08, we are expected to subtract SRCREG1 from SRCREG2 instead of SRCREG2 from SRCREG1. Therefore, we implemented another circuit to overcome this issue as shown in projects parts part. This part was the hardest part of the project because it was very complex and time consuming. Even though we spent lots of time and energy on this part, we successfully finished this part with the power of collaboration.

In 0x0B operation, our goal is firstly increment SP, then loading M[SP] to the Rx (SP <- SP+1, Rx <- M[SP]). In this operation, firstly we are setting RegSelB and FunSelB in order to increment SP, then giving a clock, then we are setting OutDSel, Read, MuxASel, RegSelA and FunSelA in order to load M[SP] to the Rx as mentioned in results part. In this part, we could not decide if we should implement both of these operations in one clock cycle or consecutively but then we realized that we can not implement it in just one clock cycle. Therefore, we implemented this operation in two clock cycles and finished this part successfully.

In 0x0C operation, our goal is firstly loading Rx to the M[SP], then decrement SP (M[SP] <- Rx, SP <- SP-1). In this operation, firstly we are setting OutDSel, OutASel, FunSelALU and Load in order to load Rx to the M[SP], then giving a clock, then we are setting RegSelB and FunSelB in order to decrement SP as mentioned in results part. In this part, we could not decide if we should implement both of these operations in one clock cycle or consecutively but then we realized that we can not implement it in just one clock cycle. Therefore, we implemented this operation in two clock cycles and finished this part successfully.

In 0x0D and 0x0E operations, our goal is implementing increment and decrement operations (For 0x0D: DESTREG <- SRCREG1 + 1 ; For 0x0E: DESTREG <- SRCREG1 - 1). Even though our goal is not same for this operations, the implementation methods of these operations are quite similar. The only difference is setting FunSelA and FunSelB. Therefore, we implemented these operations together. However, since both of DESTREG and SRCREG1 can be choosen from Register File or Address Register File, there are four different possibilities in these operations. As we explained in previous paragraphs, we used the same logic and we did our implementations separately for each possibility as mentioned in results part. However, we faced with an issue when our DESTREG and SRCREG1 is the same register. To solve this issue, we desgined a circuit with 4 XNOR

gates in order to check if DESTREG and SRCREG1 is same or not as mentioned in results part. Solving this problem was challenging but we finished this part successfully.

In 0x0F operation, our goal is loading value to the PC if Z = 0 (IF Z = 0 THEN PC <- Value). In this operation, we did almost same implementations we did in 0x00 operation (PC <- Value). The only difference is, when determining the state and gate, Control Unit input Z value connected with NOT gate in order to skip this if Z flag is 1. Since this operation was quite similar to the operation 0x00, we finished this part easily and successfully.

5 CONCLUSION

In this second project, we have designed a control unit for the circuit we have designed in the first project. The outputs of this control unit control the inputs of the main circuit. With combination of these two projects, we have designed a basic computer and we have learned how a basic computer works. At the beginning of the project we thought that this project is extremely hard because connecting every input of the first project to the outputs of the control unit one-by-one depending on the operations and calculating all of the possibilities of DEST, SRC1 and SRC2 registers seemed very complex. But then, we made tables for each operation and wrote down every possibilities and every inputs to be changed and to be connected. With this way we saw that there exist many similar implementations we should do and we saw that our job is get much more easier. Beyond that, we faced with other small errors but we have overcome these issues as a group. Eventually, we really get used of it and we thought that even though this project seems very hard and complex, it is also quite enjoyable when you totally understood its logic.

REFERENCES