ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 2

DUE DATE : 09.06.2021

GROUP NO : G55

GROUP MEMBERS:

150190710 : Serra Bozkurt

150190063 : Gökalp Akartepe

150190024 : Ahmet Furkan Kavraz

Contents

1 INTRODUCTION

In this homework, we are asked to implement a control unit circuit, so that instead of assigning selectors/enables/other inputs manually, we let the circuit decide which operation to do while the given opcodes operate. We had different types of registers, register file, address register file, Arithmetic Logic Unit and a circuit consists of these components with the same clock signal. We replaced each input in the main circuit with a tunnel, and used those tunnels to retrieve appropriate data from control unit.

For these implementations, we have reviewed our previous implementations of registers, register files and ALU.

After all, we implemented inputs of each modules one by one according to the expectations in the homework documentation. See Figure 1.

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 1: OPCODE field & Symbols for operations & their descriptions

All inputs are made according to the tables in the given documentation.

Then, we tested these circuits with different test cases. Finally, we checked the results to see if the outputs match our expectations.

2 PROJECT PARTS

2.1 FETCH AND DECODE

Fetching the operation from memory and decoding it requires 2 clock cycles, at the beginning of every operation. The reason for that is to first fill the LSB's of IR, and then to fill the MSB's of the IR. So during these operations, the FunSel of IR should be on the load mode.

2.1.1 T0

During the first clock signal, we will fetch the LSB's of IR, from memory, and load them to IR. For these, we should have the following properties about circuit inputs:

- We should have the OutDSel of ARF to be 00, so that we can retrieve PC data and use it as a memory pointer, to start the operations. See Figure 24 for visual explanation.
- We should have the FunSel of ARF to be 01, so that we can increment the PC after the first clock cycle, to have it point the the right next to the location it points before. See Figure 22 for visual explanation.
- We should have the RegSel of ARF to be 011, so that we can enable just the the PC during the operation, and increment it afterwards. If any other registers in the ARF are needed, we will activate them later by changing the RegSel and enabling them. See Figure 25 for visual explanation.
- We should have the SELECT of the Memory to be 1, so that it points to where the PC sends it. See Figure 27 for visual explanation.
- We should have the LOAD of the Memory to be 1, so that it outputs the data it currently points (M[PC]). See Figure 27 for visual explanation.
- We should have the Enable of IR to be 1, so that we can activate the IR. See Figure 16 for visual explanation.
- We should have the FunSel of IR to be 10, so that we can load the data retrieved from memory to IR's LSB's. See Figure 16 for visual explanation.
- We should have the L/H of IR to be 0, so that we can enable the LSB's of IR first. See Figure 16 for visual explanation.

2.1.2 T1

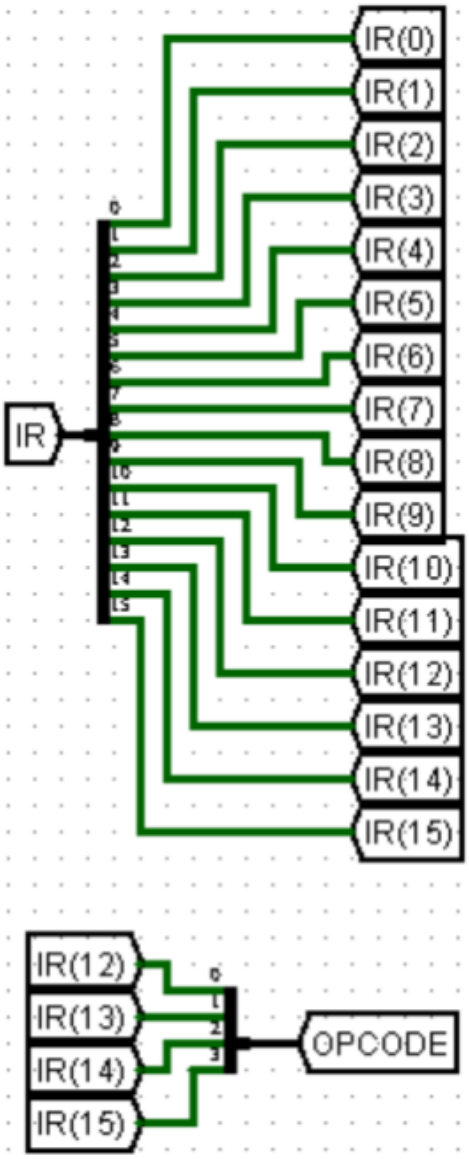
During the second clock signal, we will fetch the MSB's of IR, from memory, and load them to IR. For these, we should have the following properties about circuit inputs:

- We should have the OutDSel of ARF to be 00, so that we can retrieve PC data and use it as a memory pointer, to continue the operations. See Figure 24 for visual explanation.
- We should have the FunSel of ARF to be 01, so that we can increment the PC after the second clock cycle, to have it point the the right next to the location it points before. See Figure 22 for visual explanation.
- We should have the RegSel of ARF to be 011, so that we can enable just the the PC during the operation, and increment it afterwards. If any other registers in the ARF are needed, we will activate them later by changing the RegSel and enabling them. See Figure 25 for visual explanation.
- We should have the SELECT of the Memory to be 1, so that it points to where the PC sends it. See Figure 27 for visual explanation.
- We should have the LOAD of the Memory to be 1, so that it outputs the data it currently points (M[PC]). See Figure 27 for visual explanation.
- We should have the Enable of IR to be 1, so that we can activate the IR. See Figure 16 for visual explanation.
- We should have the FunSel of IR to be 10, so that we can load the data retrieved from memory to IR's MSB's. See Figure 16 for visual explanation.
- We should have the L/H of IR to be 1, so that we can enable the MSB's of IR first. See Figure 16 for visual explanation.

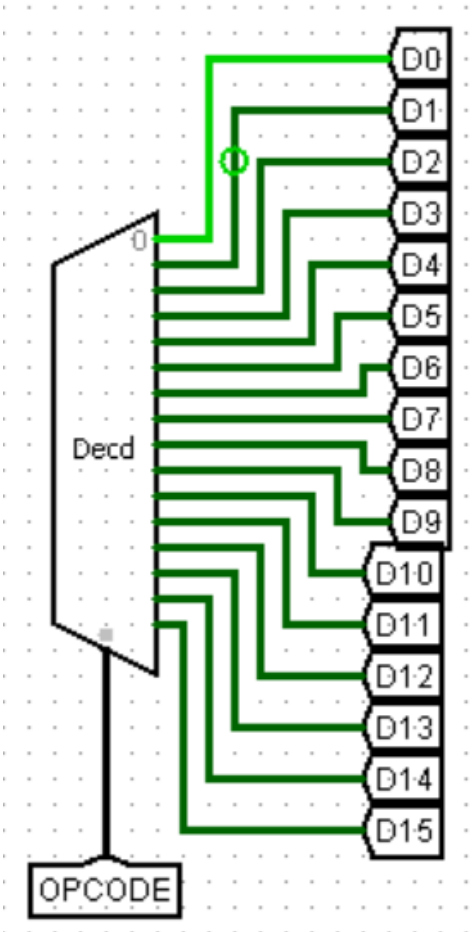
2.1.3 Decode

Then, we get the Instruction Register's all the bits to the control unit. Then, we splitted them according to our needs. See Figure 2a. After obtaining IR's bits, we picked the OPCODE bits from it and decoded the opcode bits into tunnels, which we will later use in out implementations as inputs specifying the operation. See Figure 2b for visual explanation, and we're done fetching and decoding.

After IR splitting, we decided to keep in hand the useful type of inputs, according to different address-referenced types, given to us in the pdfs. To different address reference types and their related Instruction Register bits, see Figure 3 and Figure 4.



(a) Instruction Register bits splitted into regarding tunnels



(b) OPCODE bits decoded to obtain the related instructions

OPCODE (4-bit)	0 (1-bit)	ADDRESSING MODE (1-bit)	REGSEL (2-bit)	ADDRESS (8-bit)

Figure 3: IR bits splitted with address reference

OPCODE (4-bit)	DESTREG (4-bit)	SRCREG1 (4-bit)	SRCREG2 (4-bit)

Figure 4: IR bits splitted without address reference

So we splitted IR bits according to this criterion. We first implemented the splitting with address reference, and then we implemented the splitting without address reference.

So from these split operations we obtained the tunnels listed above:

- Instructions with address reference:
  - Addressing Mode
  - ADDRESS
  - RegSel

  See Figure 5 for them.

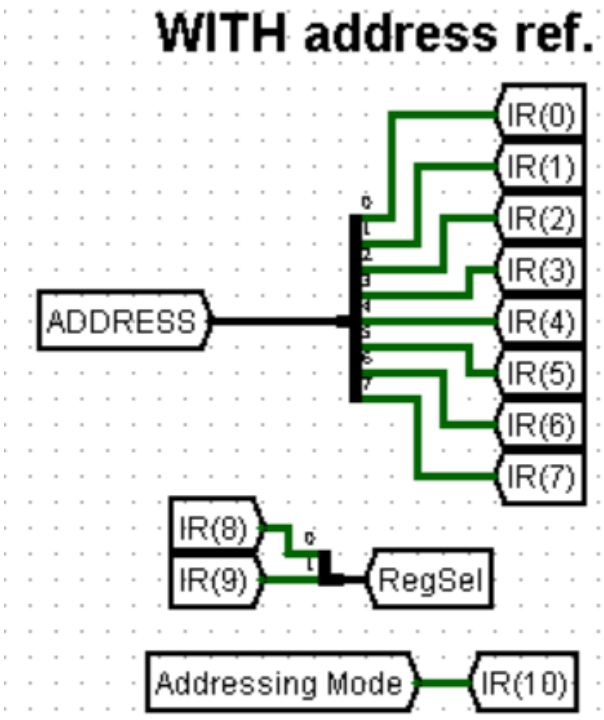


Figure 5: Instruction Register bits splitted for implementations WITH address reference, using Figure 3 and Figure 11.

REGSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

Figure 6: RegSel bits and their corresponding registers

- Instructions without address reference:
  - DESTREG
  - SRCREG1
  - SRCREG2

  See Figure 7 for them.

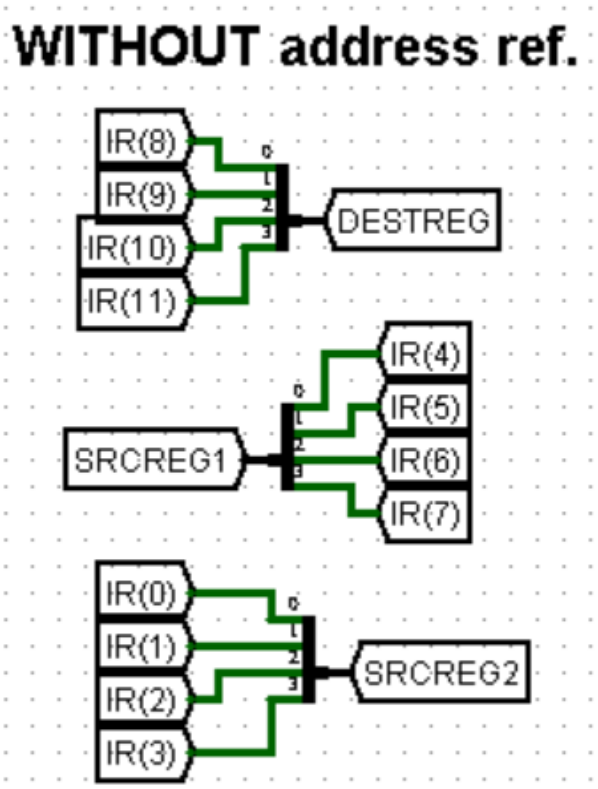
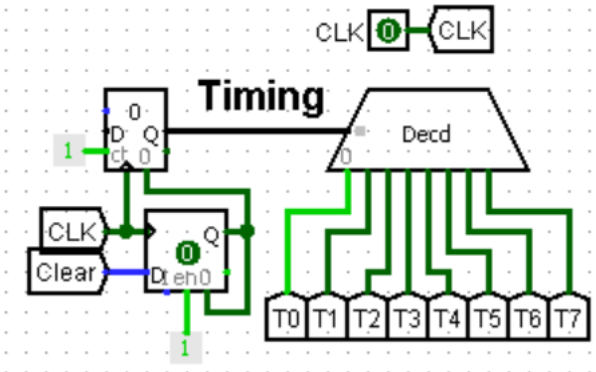


Figure 7: Instruction Register bits splitted for implementations WITHOUT address reference, using Figure 4.

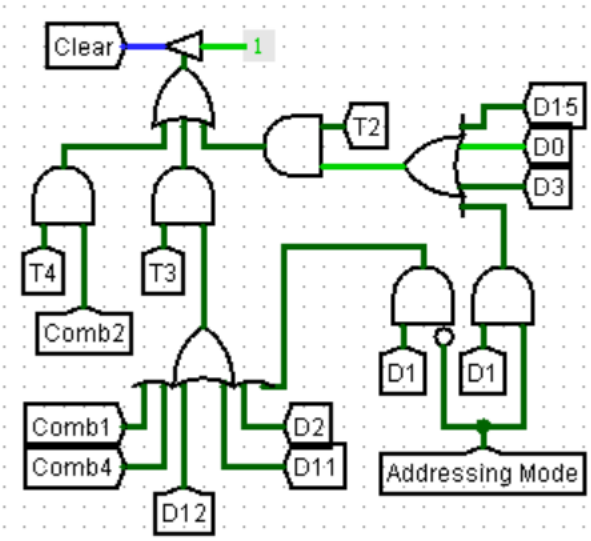
DESTREG/SRCREG1/SRCREG2	REGISTER
0000	PC
0001	PC
0010	AR
0011	SP
0100	R1
0101	R2
0110	R3
0111	R4

Figure 8: DESTREG/SRCREG1/SRCREG2 and their corresponding registers

Then, we implemented the timing part, which helped us during the clock signal management. We also implemented the Clear signal, and activated it by using different ending times of the inputs. See See Figure 9a and Figure 9b. We will dive deep into the explanations of the timing signals and clearing them later in the report.



(a) Time signals we created



(b) Clear signal activation requirements

We grouped our different opcodes into some combinational sets, regarding to their implementation similarity or followed path similarity, as shown in the Figure 10. We will explain why we thought they are similar, later in the report parts.

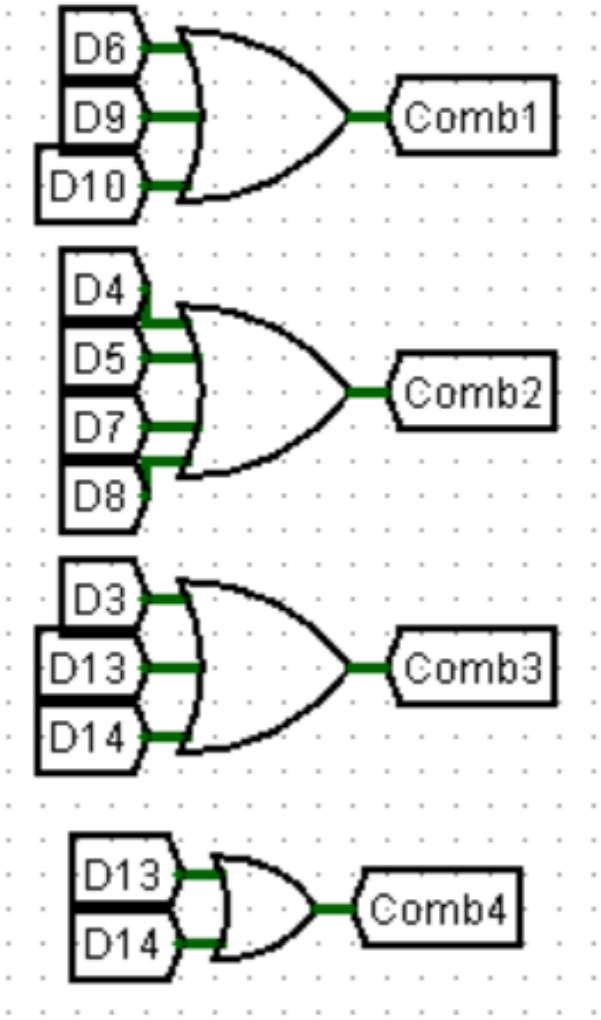


Figure 10: Combinations of Lines

2.2 D0

After fetching at T0 and decoding at T1, during the operation 0x00, we are trying to assign the value to PC. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to Figure 11 from pdf, the addressing mode is immediate, so the Value stands for ADDRESS Field. So we will load the value in the ADDRESS Field to PC.

1. Since we already have the ADDRESS Field loaded to IR(during T0 AND T1), the only thing we need to do is to set MuxBSel in a way to make ARF access the ADDRESS Field. To do that, we will assign the MuxBSel to 10 (See Figure 29).

2. Then, we should enable the PC to fetch the data came from ADDRESS Field via Mux B. To do that:

   - Set Funsel of ARF to 10 (See Figure 22).
   - Set RegSel of ARF to 011 (See Figure 25).

2.3 D1

After fetching at T0 and decoding at T1, during the operation 0x01, we are trying to assign the value to Rx. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to Figure 11 from pdf, we have two different addressing modes. We considered each:

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

Figure 11: Addressing mode part of IR bits from pdf

2.3.1 Addressing Mode: DIRECT

If addressing mode is DIRECT, we will assign the value in M[AR] to Rx. To do that, we will continue along the following path:

1. First, fetch AR from ARF. To do that, we will set OutDSel of ARF to 10, which will load AR to memory (See Figure 24).
2. Then, load the value from M[AR] to Memory output. To do that, we will:

   - Set the SEL selector of memory to 1 - to go to location AR's value points (See Figure 27).
   - Set the LOAD selector of memory to 1 - to load the current selector's value to output (See Figure 27).

3. After these, we should set the Mux A in such a way that M[AR] becomes accessible by Register File. To do that, we will assign the MuxASel to 01 (See Figure 28).
4. Finally, we should enable the Register File's Rx to fetch the data came from M[AR] to Mux A. To do that:

- Set Funsel of Register File to 10 (See Figure 17).
- Set RegSel of Register File to decoded output of RegSel. It will decide what Rx will be (See Figure 20).

2.3.2 Addressing Mode: IMMEDIATE

If addressing mode is IMMEDIATE, we will assign the value in ADDRESS Field to Rx.

1. Since we already have the ADDRESS Field loaded to IR(during T0 AND T1), the only thing we need to do is to set MuxASel in a way to make Register File access the ADDRESS Field. To do that, we will assign the MuxASel to 10 (See Figure 28).
2. Then, we should enable the Register File's Rx to fetch the data came from ADDRESS Field to Mux A. To do that:
   - Set Funsel of Register File to 10 (See Figure 17).
   - Set RegSel of Register File to decoded output of RegSel. It will decide what Rx will be (See Figure 20).

2.4 D2

After fetching at T0 and decoding at T1, during the operation 0x02, we are trying to assign the Rx to Value. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to Figure 11 from pdf, the addressing mode is direct, so the Value stands for M[AR]. So we will load the value in the Rx to M[AR].

1. First, we should give the value in the Register File to ALU, since its the only way between Memory and Register File. So we set the OutASel to RegSel's decoded value, to let it fetch the data from correct Register-Rx (See Figure 18).
2. Then, we should let the data pass through ALU without any changes, so we set ALU'S FunSel to 0000, which means directly get value from input-A and give it as OutAlu (See Figure 26).
3. Then, we should make the arrangements with the memory. To do that:
   - Set OutDsel of ARF to 10 - to get AR as a pointer to memory (See Figure 24).

- Set STORE of Memory to 1 - to store the data from Rx to M[AR] (See Figure 27).
- Set SELECT of Memory to 1 - to select the appropriate place to store the data which is coming from Rx via ALU to M[AR] (See Figure 27).

2.5 COMB3(D3-D13-D14)

After fetching at T0 and decoding at T1, during the operation 0x03, we are trying to assign the SRCREG1 to DESTREG. And during the operations 0x0D & 0x0E, we are assigning the SRCREG1 to DESTREG, too. But just with a little additional clock cycle, which will activate the FunSel of DESTREG, so we first will assign the data to DESTREG and then either increment or decrement it in its place. That's why we combined these operations.

Combination Reason:

$D3 + 1\_clock\_cycle\_with\_FunSel = D13\_or\_D14$

While doing that, we should first decide whether the DESTREG and/or the SRCREG1 is on the ARF or Register File, to find out what the path data will follow is. According to Figure 8 from pdf, there are 4 different cases that changes the data path.

1. DESTREG in Register File & SRCREG1 in Register File.
2. DESTREG in Register File & SRCREG1 in ARF.
3. DESTREG in ARF & SRCREG1 in Register File.
4. DESTREG in ARF & SRCREG1 in ARF.

2.5.1 Case-1: DESTREG in Register File & SRCREG1 in Register File

The path that the data will follow from SRCREG1 to DESTREG in the main circuit will be as following Figure 12 when SRCREG1 is in Register File and DESTREG is in Register File.

So, to arrange this path, we will have the following steps:

1. Load the SRCREG1 to Out-A of Register File. We should make OutASel be whatever the SRCREG1's last 2 bits decoded into. But we have the requirement of SRCREG1 being in Register File. As seen in Figure 8, if SRCREG1's 2nd bit is 1, it will be in Register File. To make sure our implementations work properly, we will give this requirement to OutASel, as an input, so that it will not operate if the SRCREG1 is not in Register File. See clear explanation in Figure 18.

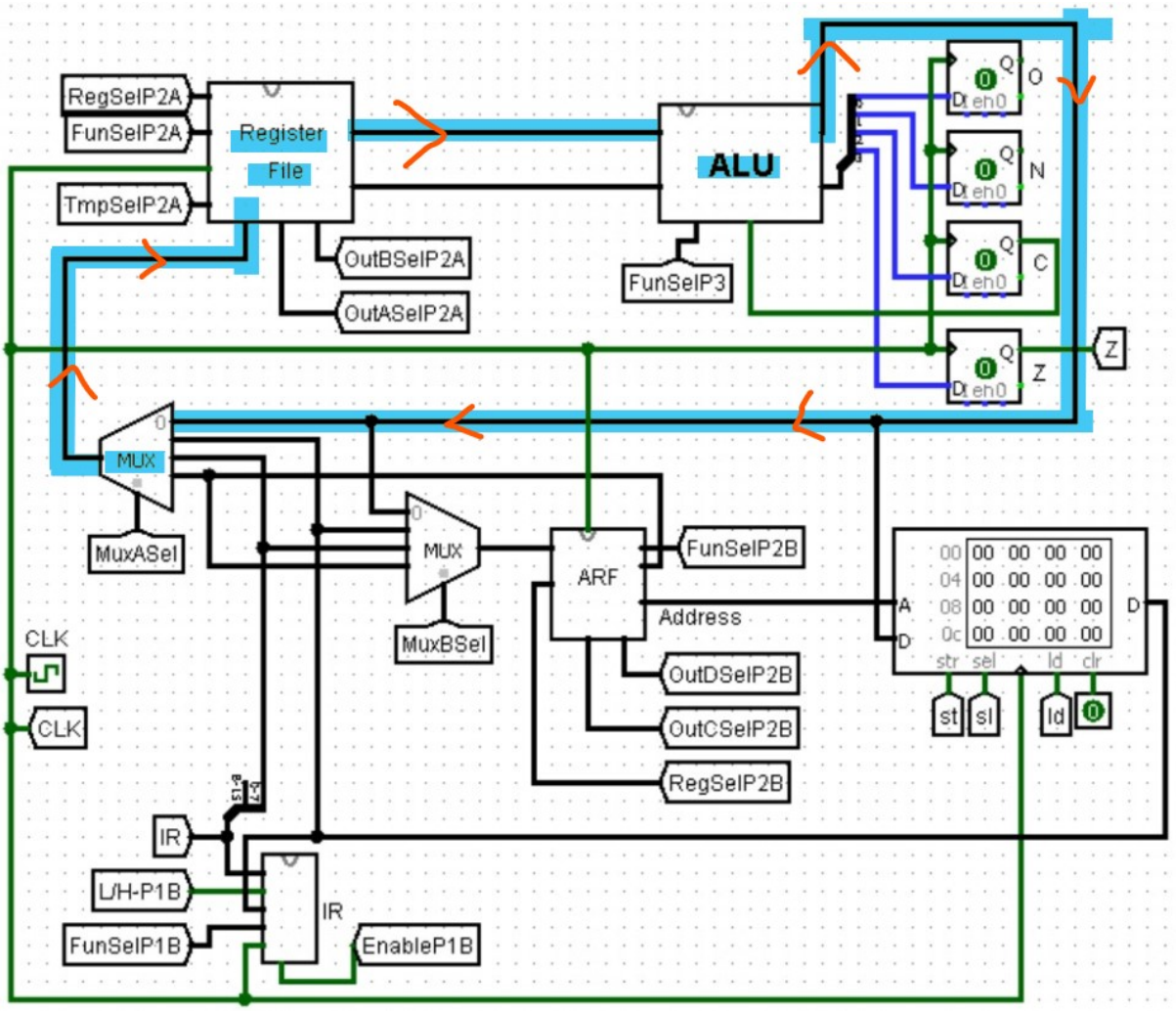


Figure 12: Path of data when SRCREG1 in Register File and DESTREG in Register File

2. Then, we should let the data pass through ALU without any changes, so we set ALU'S FunSel to 0000, which means directly get value from input-A and give it as OutAlu (See Figure 26).

3. After these, we should set the Mux A in such a way that SRCREG1 through ALU becomes accessible by Register File. To do that, we will assign the MuxASel to 00 (See Figure 28).

4. Finally, we should enable the Register File's Rx to fetch the data came from SRCREG1 to Mux A. To do that:

   - Set Funsel of Register File to 10 (See Figure 17).
   - Set RegSel of Register File to decoded output of RegSel, which also corresponds to DESTREG's least significant 2 bits. It will decide what Rx will be (See Figure 20) To ensure this implementation works only when DESTREG is in Register File, we check the DESTREG's 2nd bit. As seen in Figure 8, if DESTREG's 2nd bit is 1, it will be in Register File. We will give this requirement to RegSel of Part 2A, as an input, so that it will not operate if the DESTREG is not in Register File. See clear explanation in Figure 20.

2.5.2 Case-2: DESTREG in Register File & SRCREG1 in ARF

The path that the data will follow from SRCREG1 to DESTREG in the main circuit will be as following Figure 13 when SRCREG1 is in ARF and DESTREG is in Register File.

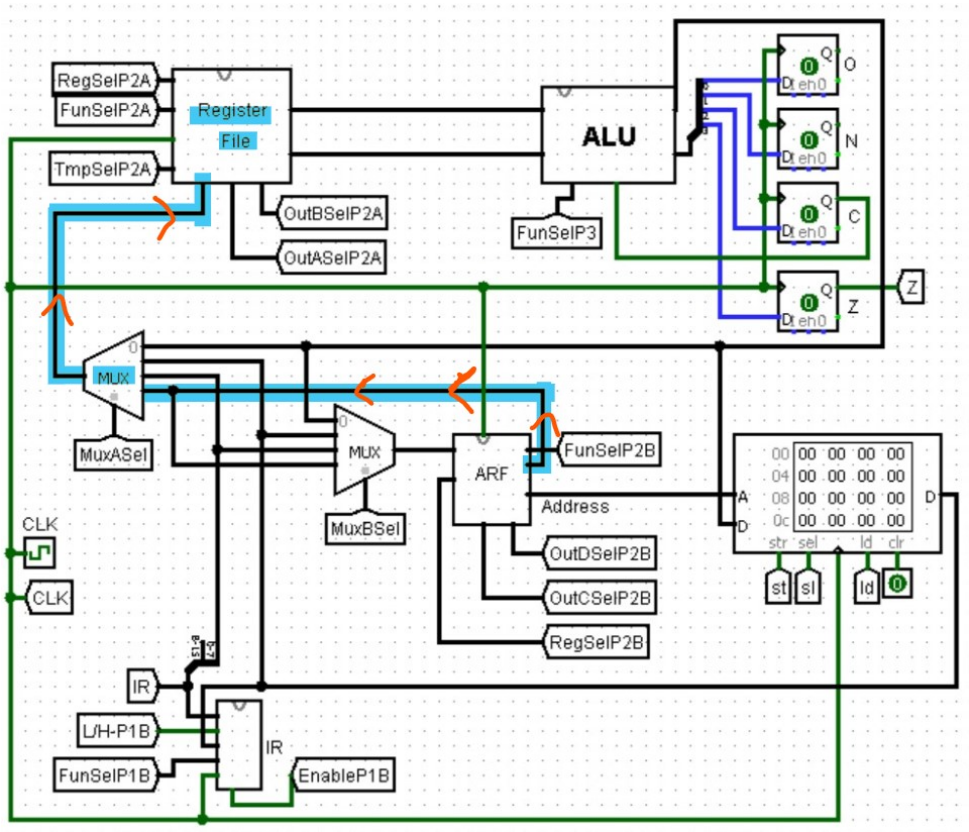


Figure 13: Path of data when SRCREG1 in ARF and DESTREG in Register File

So, to arrange this path, we will have the following steps:

1. Load the SRCREG1 to Out-C of ARF. We should make OutCSel be whatever the SRCREG1's last 2 bits decoded into. But we have the requirement of SRCREG1 being in ARF. As seen in Figure 8, if SRCREG1's 2nd bit is 0, it will be in ARF. To make sure our implementations work properly, we will give this requirement to OutCSel, as an input, so that it will not operate if the SRCREG1 is not in ARF. See clear explanation in Figure 23.

2. After these, we should set the Mux A in such a way that SRCREG1 becomes accessible by Register File. To do that, we will assign the MuxASel to 11 (See Figure 28).

3. Finally, we should enable the Register File's Rx to fetch the data came from SRCREG1 to Mux A. To do that:

   - Set Funsel of Register File to 10 (See Figure 17).
   - Set RegSel of Register File to decoded output of RegSel, which also corresponds to DESTREG's least significant 2 bits. It will decide what Rx will be (See Figure 20) To ensure this implementation works only when DESTREG is in Register File, we check the DESTREG's 2nd bit. As seen in Figure 8, if DESTREG's 2nd bit is 1, it will be in Register File. We will give this requirement to RegSel of Part 2A, as an input, so that it will not operate if the DESTREG is not in Register File. See clear explanation in Figure 20.

2.5.3 Case-3: DESTREG in ARF & SRCREG1 in Register File

The path that the data will follow from SRCREG1 to DESTREG in the main circuit will be as following Figure 14 when SRCREG1 is in Register File and DESTREG is in ARF.

So, to arrange this path, we will have the following steps:

1. Load the SRCREG1 to Out-A of Register File. We should make OutASel be whatever the SRCREG1's last 2 bits decoded into. But we have the requirement of SRCREG1 being in Register File. As seen in Figure 8, if SRCREG1's 2nd bit is 1, it will be in Register File. To make sure our implementations work properly, we will give this requirement to OutASel, as an input, so that it will not operate if the SRCREG1 is not in Register File. See clear explanation in Figure 18.

2. Then, we should let the data pass through ALU without any changes, so we set ALU'S FunSel to 0000, which means directly get value from input-A and give it as OutAlu (See Figure 26).

3. After these, we should set the Mux B in such a way that SRCREG1 through ALU becomes accessible by ARF. To do that, we will assign the MuxBSel to 00 (See Figure 29).

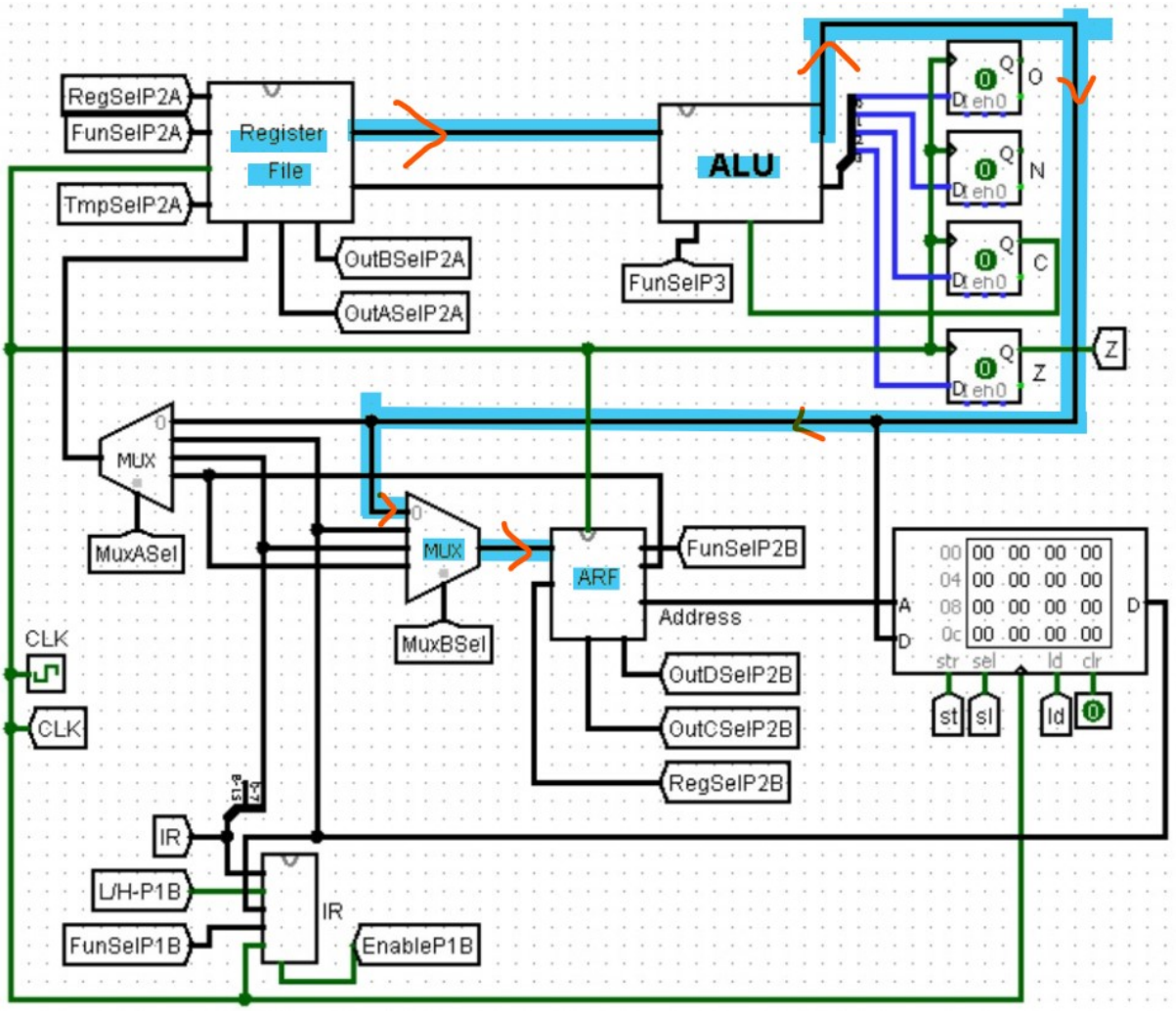


Figure 14: Path of data when SRCREG1 in Register File and DESTREG in ARF

4. Finally, we should enable the ARF's corresponding register to fetch the data came from SRCREG1 to Mux B. To do that:

   - Set Funsel of ARF to 10 (See Figure 22).
   - Set RegSel of ARF to decoded output of RegSel, which also corresponds to DESTREG's least significant 2 bits. It will decide what the register will be (See Figure 25) To ensure this implementation works only when DESTREG is

in ARF, we check the DESTREG's 2nd bit. As seen in Figure 8, if DESTREG's 2nd bit is 0, it will be in Register File. We will give this requirement to RegSel of Part 2B, as an input, so that it will not operate if the DESTREG is not in ARF. See clear explanation in Figure 25.

2.5.4 Case-4: DESTREG in ARF & SRCREG1 in ARF

The path that the data will follow from SRCREG1 to DESTREG in the main circuit will be as following Figure 15 when SRCREG1 is in ARF and DESTREG is in ARF.

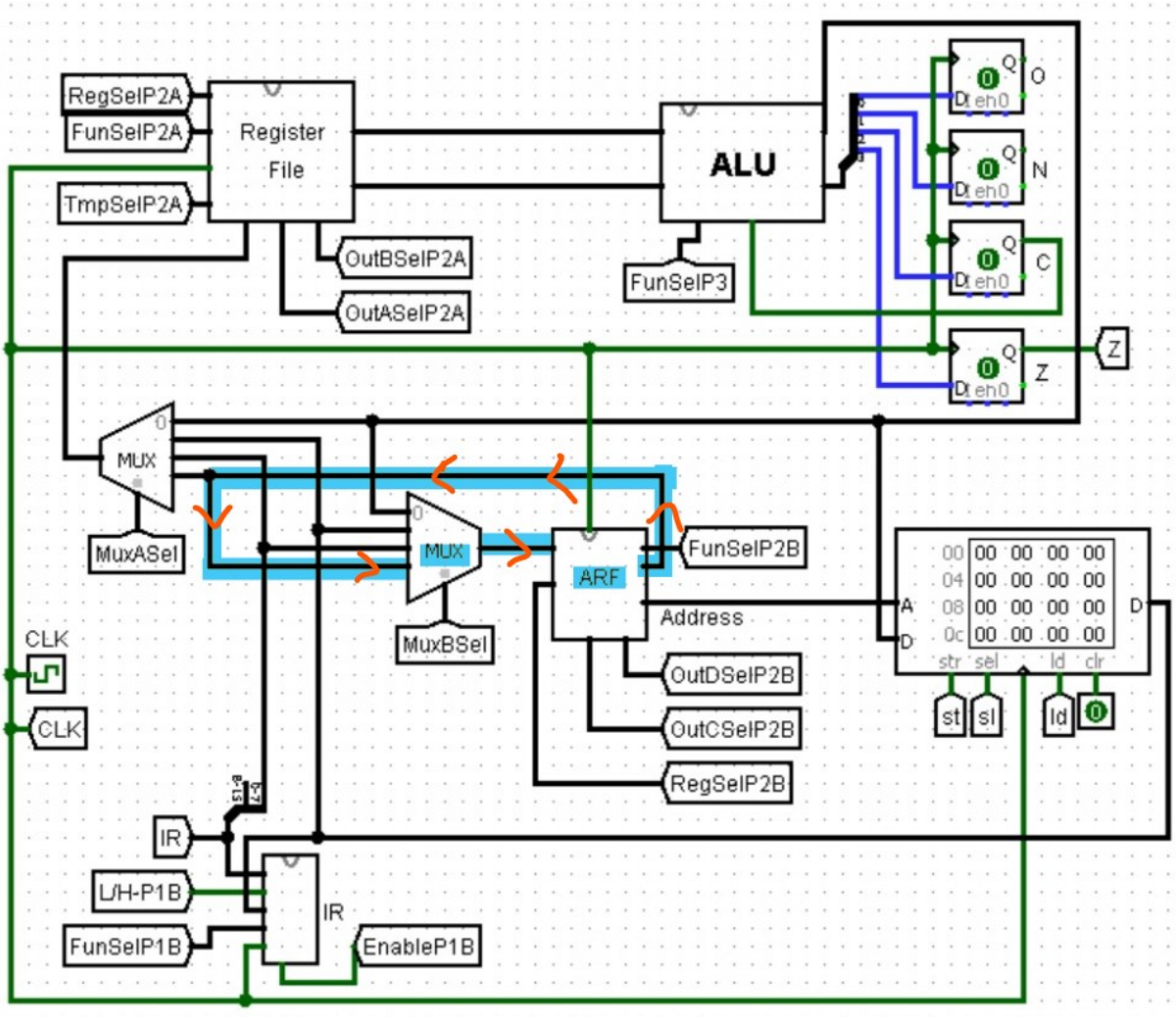


Figure 15: Path of data when SRCREG1 in ARF and DESTREG in ARF

So, to arrange this path, we will have the following steps:

1. Load the SRCREG1 to Out-C of ARF. We should make OutCSel be whatever the SRCREG1's last 2 bits decoded into. But we have the requirement of SRCREG1

being in ARF. As seen in Figure 8, if SRCREG1's 2nd bit is 0, it will be in ARF. To make sure our implementations work properly, we will give this requirement to OutCSel, as an input, so that it will not operate if the SRCREG1 is not in ARF. See clear explanation in Figure 23.

2. After these, we should set the Mux B in such a way that SRCREG1 becomes accessible by ARF. To do that, we will assign the MuxBSel to 11 (See Figure 29).

3. Finally, we should enable the ARF's corresponding register to fetch the data came from SRCREG1 to Mux B. To do that:

   - Set Funsel of ARF to 10 (See Figure 22).
   - Set RegSel of ARF to decoded output of RegSel, which also corresponds to DESTREG's least significant 2 bits. It will decide what the register will be (See Figure 25) To ensure this implementation works only when DESTREG is in ARF, we check the DESTREG's 2nd bit. As seen in Figure 8, if DESTREG's 2nd bit is 0, it will be in ARF. We will give this requirement to RegSel of Part 2B, as an input, so that it will not operate if the DESTREG is not in ARF. See clear explanation in Figure 25.

2.5.5 D13 & D14

Until now, we managed to transfer SRCREG1 to DESTREG. All we have to do left, is to decide where DESTREG is (whether the ARF or Register File) and then give one more clock cycle with arranged FunSel to:

- increment for D13
- decrement for D14

We will follow these steps to arrange funsel of correct part:

1. To ensure this implementation works only at DESTREG's correct register in the correct part of circuit, we check the DESTREG's 2nd bit. As seen in Figure 8,

   - if DESTREG's 2nd bit is 0, it will be in ARF.
   - if DESTREG's 2nd bit is 1, it will be in Register File.

2. We will give this requirement to RegSel of Part 2B AND RegSel of Part 2A, as an input, so that it will not operate if the DESTREG is not in that corresponding place. See visual explanation in Figure 25 and Figure 20.

3. After deciding what part of circuit we will use, we will activate the corresponding DESTREG register in that part.

4. We can use decoded RegSel, which also corresponds to DESTREG's least significant 2 bits. It will decide what the register will be (See Figure 25 and Figure 20)

5. Finally, we arrange the FunSel of that part to:

   - increment for D13: FunSel = 01
   - or to decrement for D14: FunSel = 00

   (See Figure 22 and Figure 17 for visual explanation) And we're done.

2.6 COMB1(D6-D9-D10)

We grouped D6-D9-D10 into Comb1. In these operations, first we fetch at T0 and decode at T1.

Then at the T2, we take corresponding value from ARF, which is indicated with last 2 bit of SRCREG1, such as for 10 we take AR regardless and assign it to TempReg3 of most significant 2 bits. We assign this value to corresponding temporary register with the inputs:

OutCSelP2B = SrcReg1[1:0], MuxAsel = 3,
TmpSelP2A = Reverse(Not(Decode(SrcReg1[1:0]))), FunSelP2A = 10 (load).

Then at the T3, we decide if the value is from ARF or Registers. According to this info we give the proper value to ALU, make the neccessary operations and assign the output of ALU to DESTREG, such as:

The input of ALU:
OutASelP2A = { Not(SrcReg1[2]) : SrcReg1[1:0] }

Setting the value of FunSel according to given operation:
if D6: (NOT Operation)
FunSelP3 = 0010
else: (LSL ya da LSR)
FunSelP3 = 101 : IR(13)

Both multiplexers are open and getting the output of ALU:
MuxASel = 00, MuxBSel = 00

Writing the output of ALU to proper DESTREG:
if DestReg[2] == 0: (data will be written into ARF)
RegSelP2B = Reverse(Not(Decode(DestReg[1:0])))
FunSelP2B = 10
else: (data will be written into Register File)
RegSelP2A = Reverse(Not(Decode(DestReg[1:0])))
FunSelP2A = 10

2.7 COMB2(D4-D5-D7-D8)

We grouped D4-D5-D7-D8 into Comb2. In these operations, first we fetch at T0 and decode at T1.

At the T2 and T3, we take the corresponding value from ARF according to last 2 bit of SRCREG1/SRCREG2, such as for 01 we take PC regardless of most significant 2 bits. And assign this value to Temporary Register with the inputs below:

T2 - OutCSelP2B = SrcReg1[1:0], MuxAsel = 3,
TmpSelP2A = Reverse(Not(Decode(SrcReg1[1:0]))), FunSelP2A = 10 (load)
T3 - OutCSelP2B = SrcReg2[1:0], MuxAsel = 3,
TmpSelP2A = Reverse(Not(Decode(SrcReg2[1:0]))), FunSelP2A = 10 (load)

Then at T4, we decide whether the value is coming from ARF or Register, and if both of the values are coming from ARF then we take two corresponding Temporary Registers' values, if only one is coming from ARF we take one corresponding Temporary Register's and one corresponding Register's value, if none of them is from ARF then we take two of the corresponding Registers' values and give them to the ALU in order to make the necessary operations and assign the output of ALU to DESTREG,:

The inputs of ALU:
OutASelP2A = { Not(SrcReg2[2]) : SrcReg2[1:0] }
OutBSelP2A = { Not(SrcReg1[2]) : SrcReg1[1:0] }

Setting the value of FunSel according to given operation:
if D4: (AND Operation) FunSelP3 = 0111
if D5: (Or Operation) FunSelP3 = 1000
if D7: (Add Operation) FunSelP3 = 0100
if D8: (Sub Operation) FunSelP3 = 0110

Both multiplexers are open and getting the output of ALU:
MuxASel = 00, MuxBSel = 00

Writing the output of ALU to proper DESTREG:
if DestReg[2] == 0: (will be written into the ARF)
RegSelP2B = Reverse(Not(Decode(DestReg[1:0])))
FunSelP2B = 10
else: (will be written into the Register File)
RegSelP2A = Reverse(Not(Decode(DestReg[1:0])))
FunSelP2A = 10

2.8 D11

We did not grouped D11 into a combination, because it would not be useful.

As always in these operations, first we fetch and decode at T0 and T1.

And at the T2, firstly we take the value from the corresponding register of Register File and also take the value of SP from ARF. And then we give these values to the memory to store the data in the appropriate location in the memory and also set store and select inputs of memory to true.

And at the T3, we decrement the SP value in the ARF by giving the appropriate inputs given below to ARF and make SP to decrement itself.

For these operations we:

- Set ARF inputs, RegSel=100, E=1, FunSel=00, OutDSel=11, for giving the value into the OutD decrementing and the SP value.
- Set Register File inputs, RegSel to Rx value in a format that according to the tables in the documents.
- Set ALU input, FunSel = 0000 for giving the A input as OutALU.

2.9 D12

We did not grouped D12 into a combination, because it would not be useful.
As always in these operations, first we fetch and decode at T0 and T1.

And at the T2, firstly we need to increment the SP value inside the ARF according to the table's 0x0C line. We increment the SP value by giving ARF to appropriate input values and make the SP register to increment itself.

After that at the T3,we take the value of SP from ARF and also take the value from the corresponding address SP of the memory. And then we give these values to the Register File to load its register via MultiplexerA. We set this multiplexers input to give the data

to register file in order to load this data from the SP address of the memory to Register File's corresponding register.

For these operations we:

- Set ARF inputs, RegSel=110, E=1, FunSel=01, OutDSel=11, for incrementing the SP value and giving the value into the OutD.
- Set Register File inputs, RegSel to Rx value in a format that according to the tables in the documents.
- Set MuxASel input to 01 to select the data coming from the memory.

2.10 D15

After fetching at T0 and decoding at T1, during the operation 0x0F, we are trying to assign the value to PC, but in addition to that, we have a requirement: Z have to be 0.

While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to Figure 11 from pdf, the addressing mode is immediate, so the Value stands for ADDRESS Field. So we will load the value in the ADDRESS Field to PC, if Z == 0. So we can do the exact same operations made during 0x00, just with an additional Z requirement.

1. Since we already have the ADDRESS Field loaded to IR(during T0 AND T1), the only thing we need to do is to set MuxBSel in a way to make ARF access the ADDRESS Field. To do that, we will assign the MuxBSel to 10, if Z == 0 (See Figure 29).
2. Then, we should enable the PC to fetch the data came from ADDRESS Field via Mux B. To do that, **if Z == 0**:
    - Set Funsel of ARF to 10 (See Figure 22).
    - Set RegSel of ARF to 011 (See Figure 25).

3 FIGURES

3.1 Part 1B

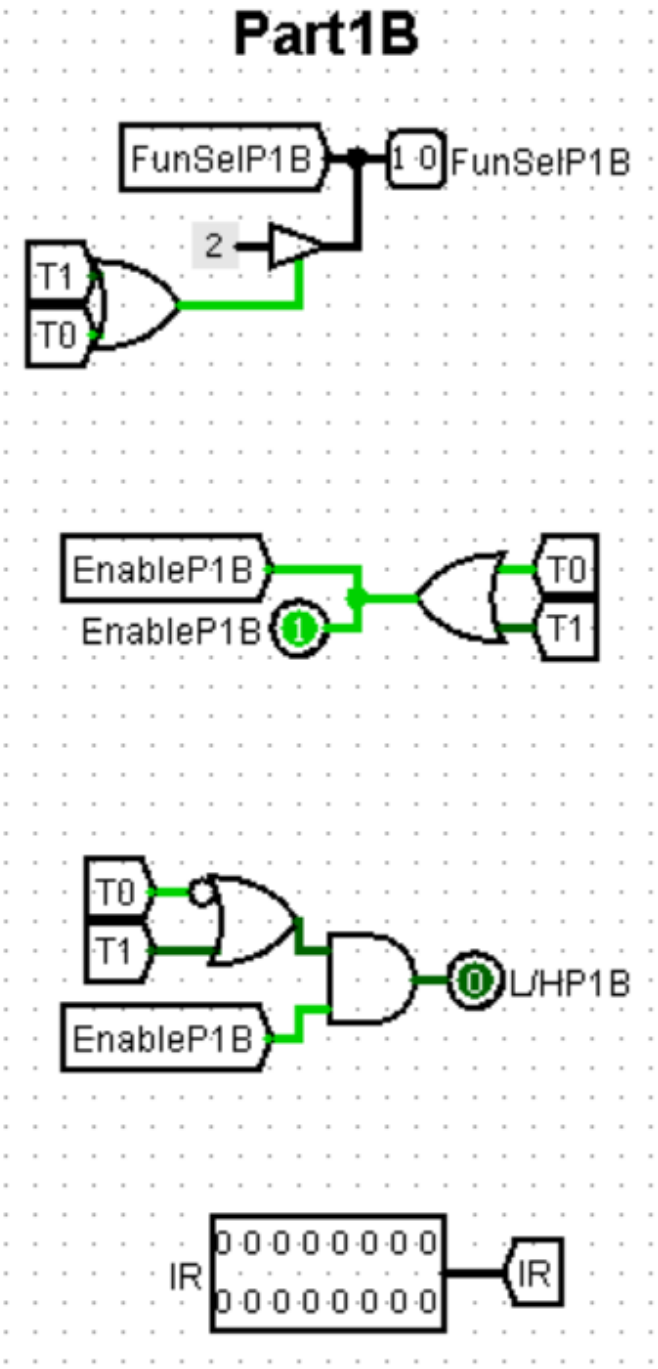


Figure 16: Instruction Register (Part 1B) components in Control Unit

3.2 Part 2A

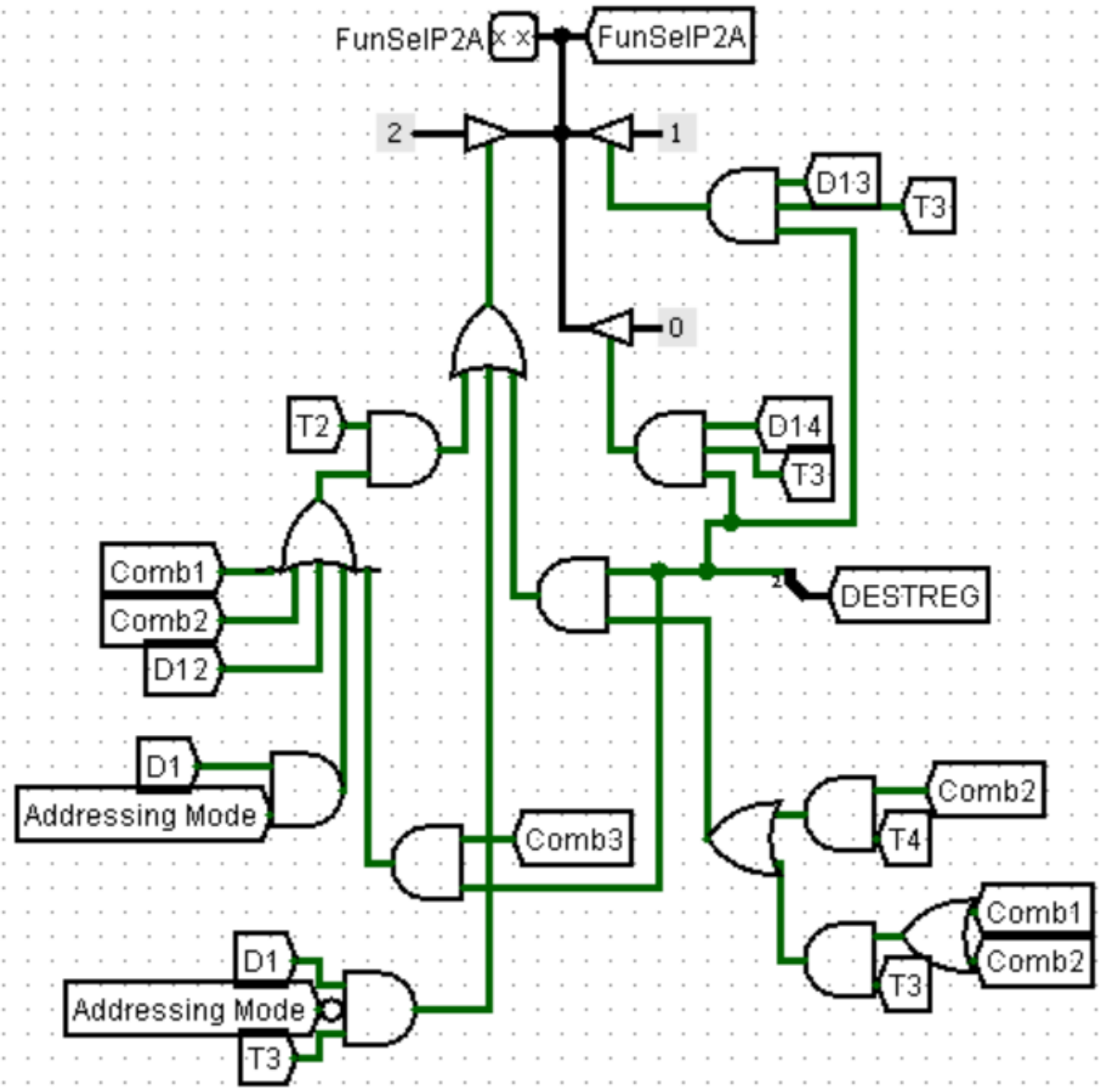


Figure 17: Part 2A FunSel in Control Unit

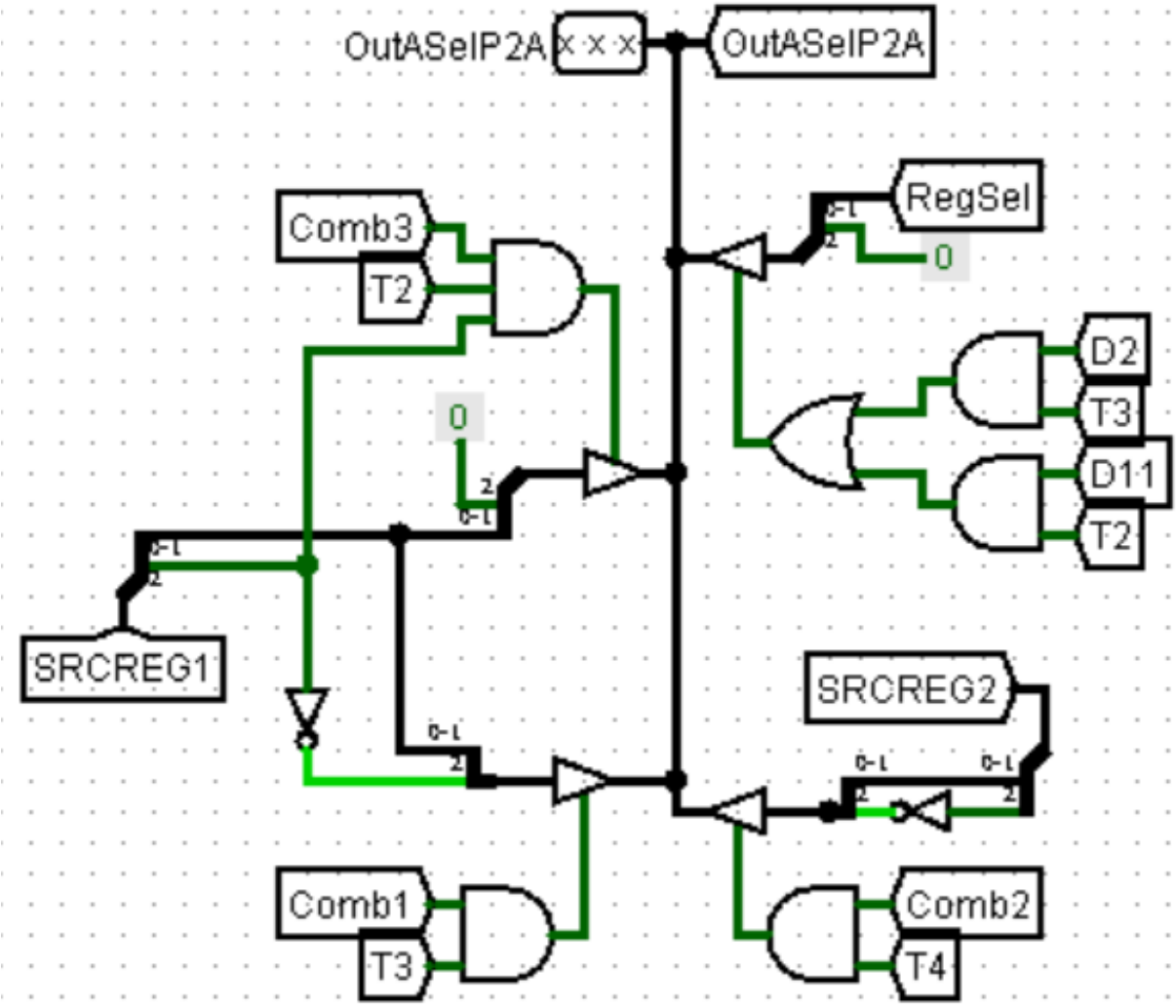


Figure 18: Part 2A OutASel in Control Unit

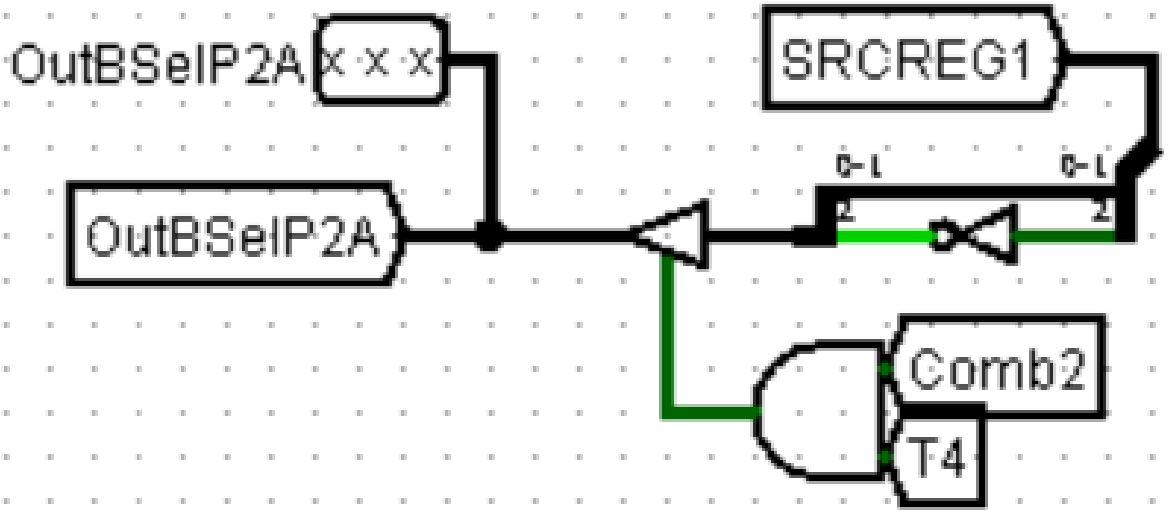


Figure 19: Part 2A OutBSel in Control Unit

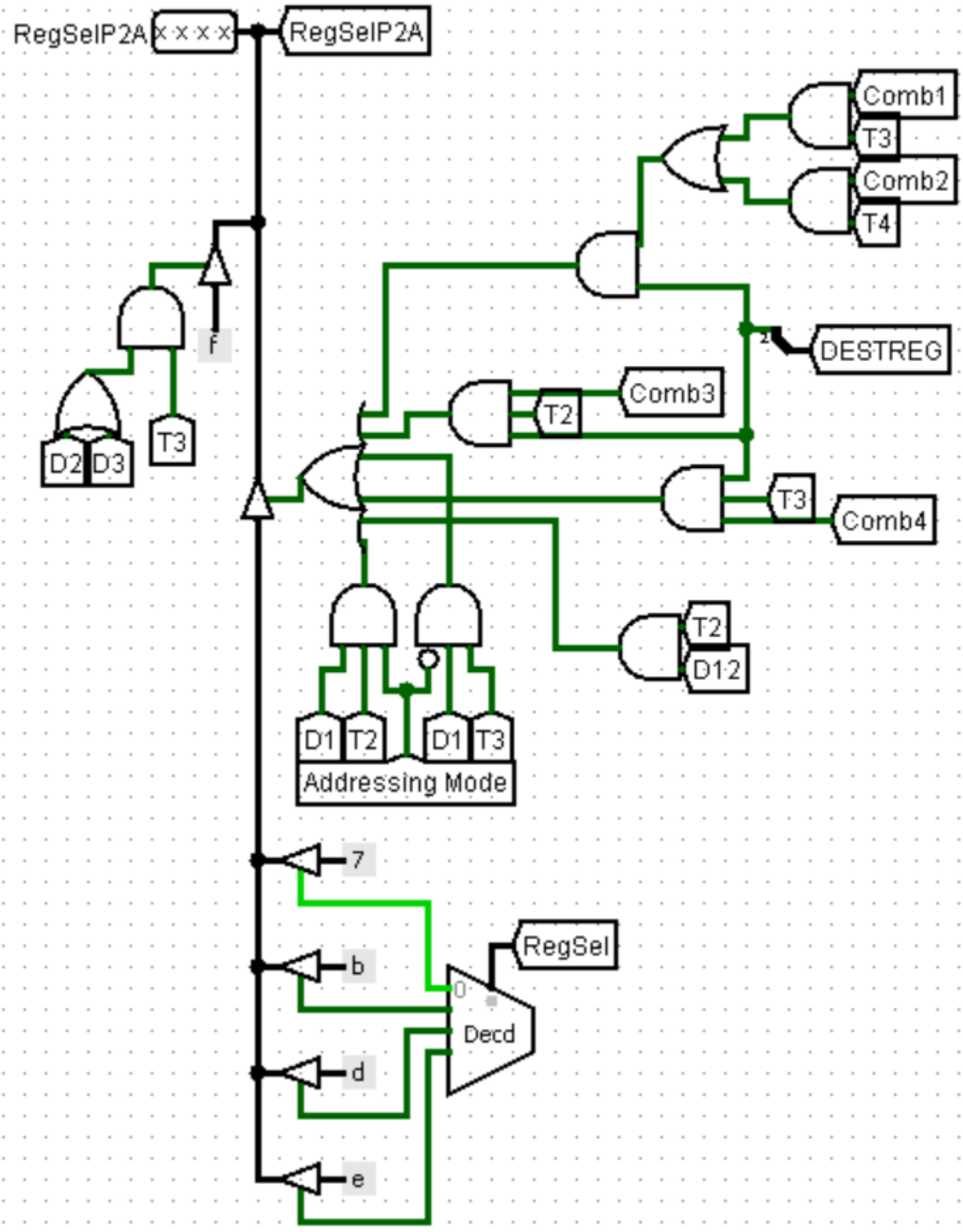


Figure 20: Part 2A RegSel in Control Unit

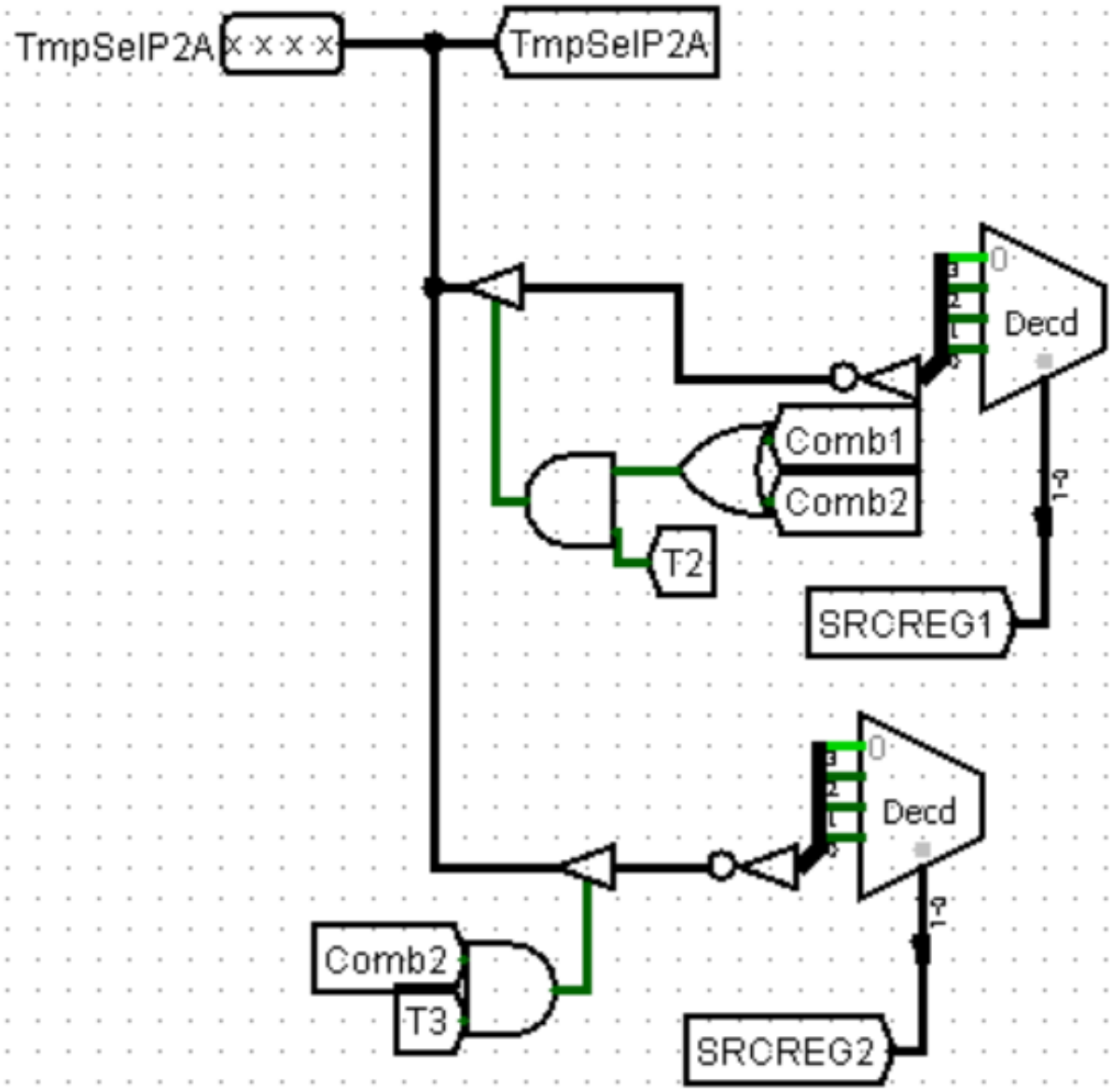


Figure 21: Part 2A TmpSel in Control Unit

3.3 Part 2B

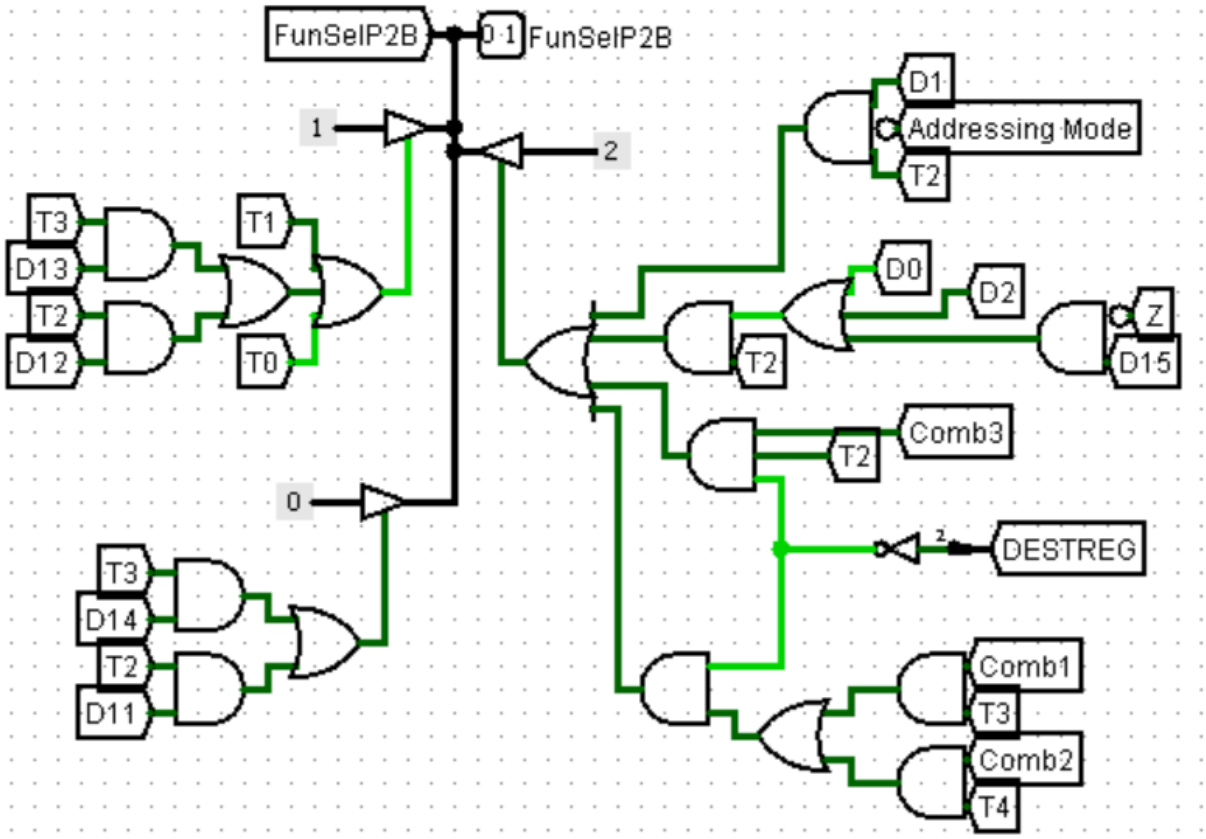


Figure 22: Part 2B FunSel in Control Unit

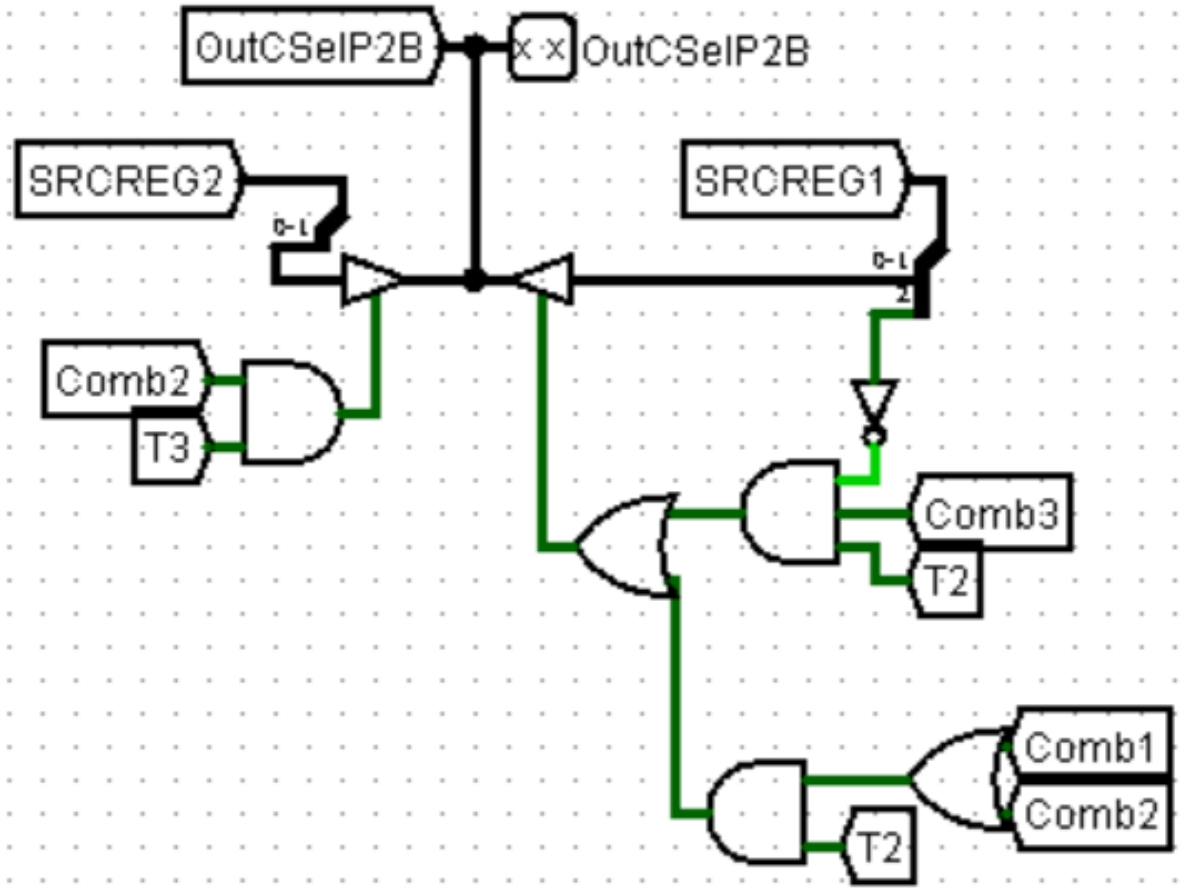


Figure 23: Part 2B OutCSel in Control Unit

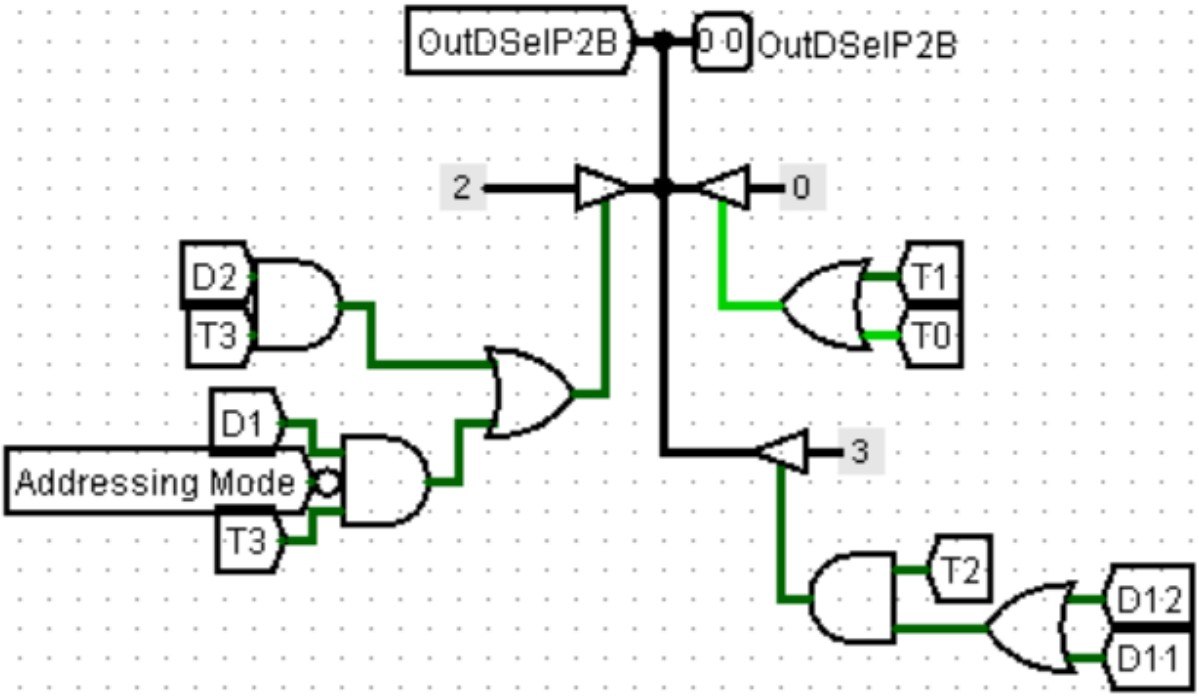


Figure 24: Part 2B OutDSel in Control Unit

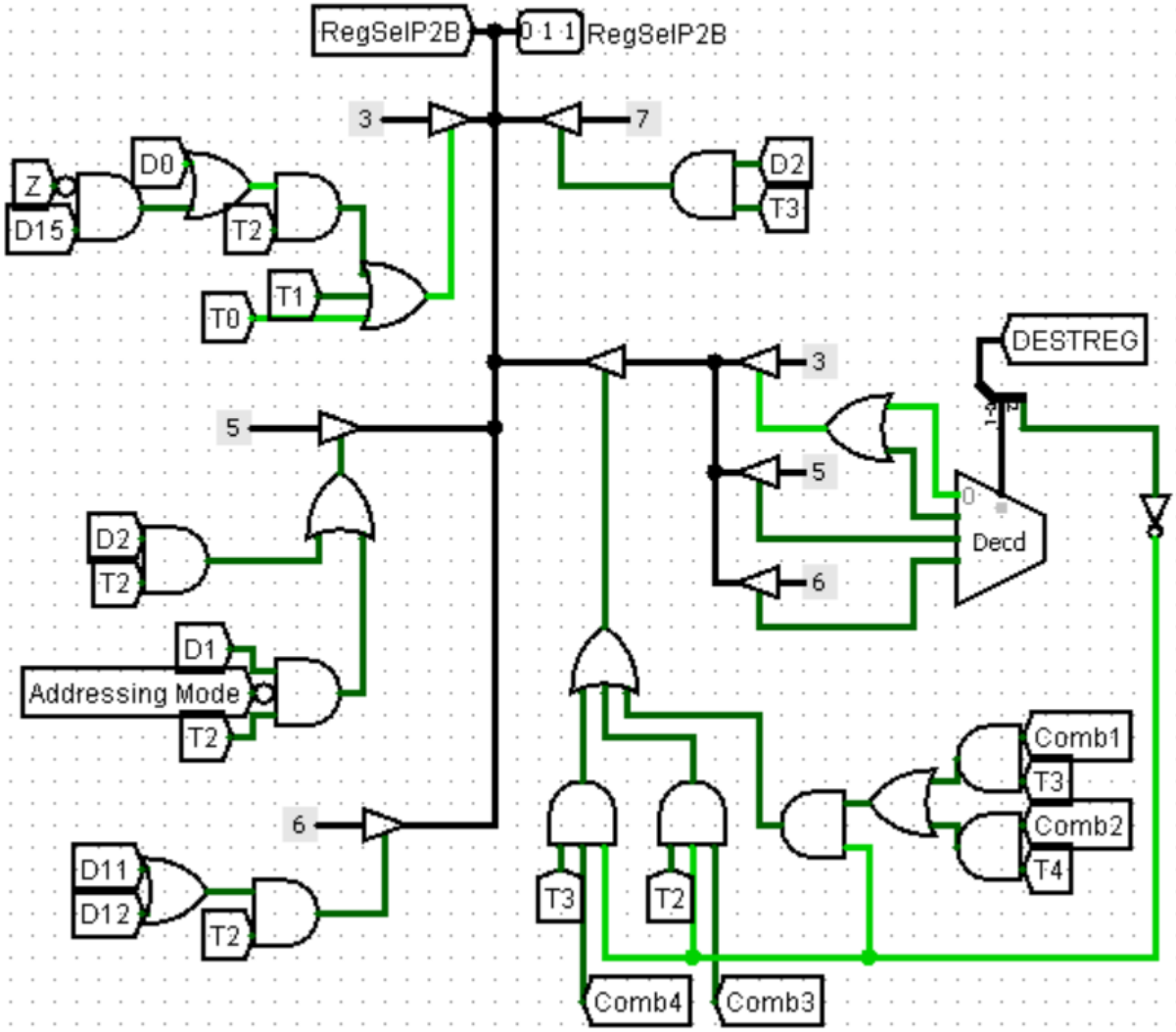


Figure 25: Part 2B RegSel in Control Unit

3.4 Part 3

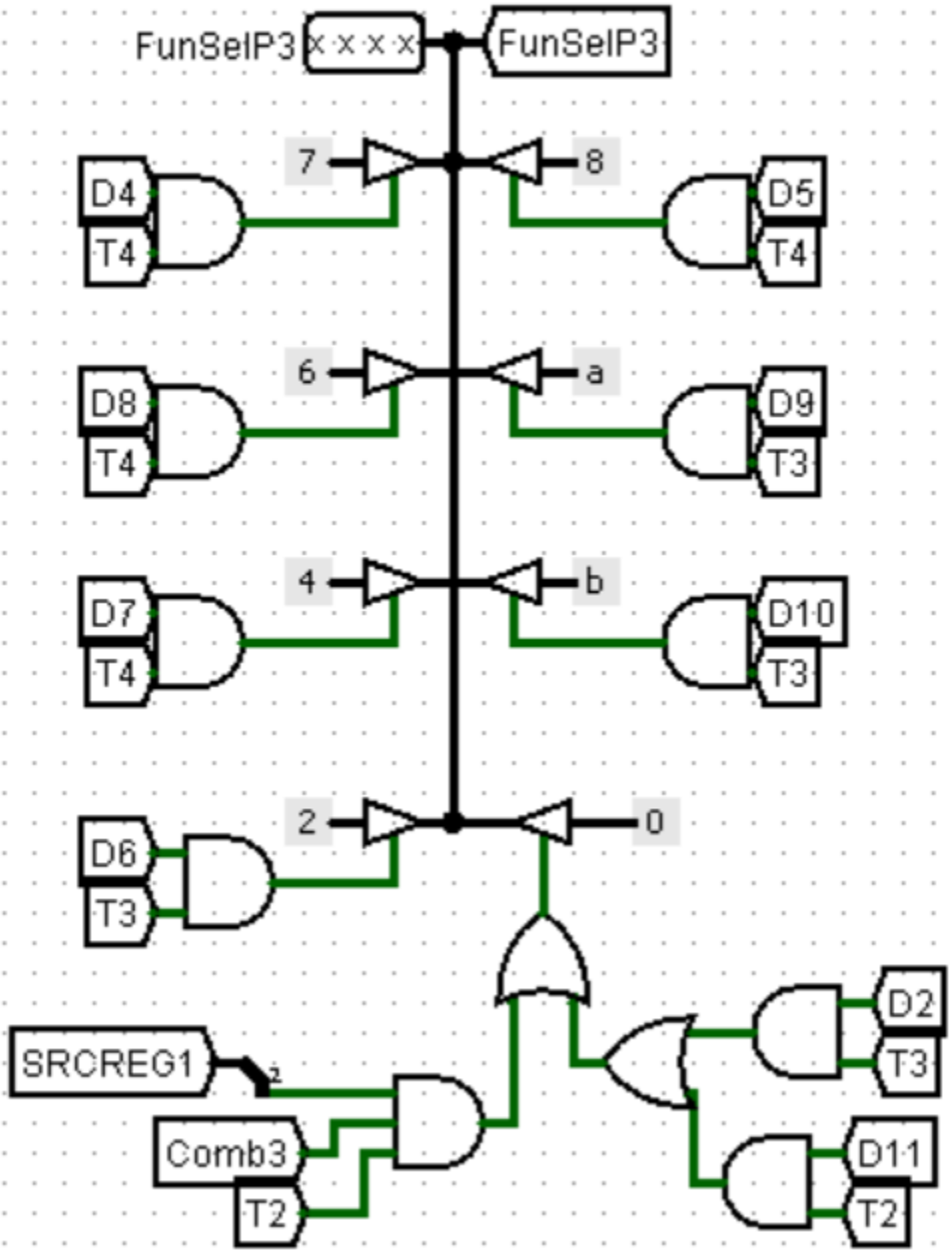


Figure 26: FunSel of ALU (Part 3) in Control Unit

3.5 Memory

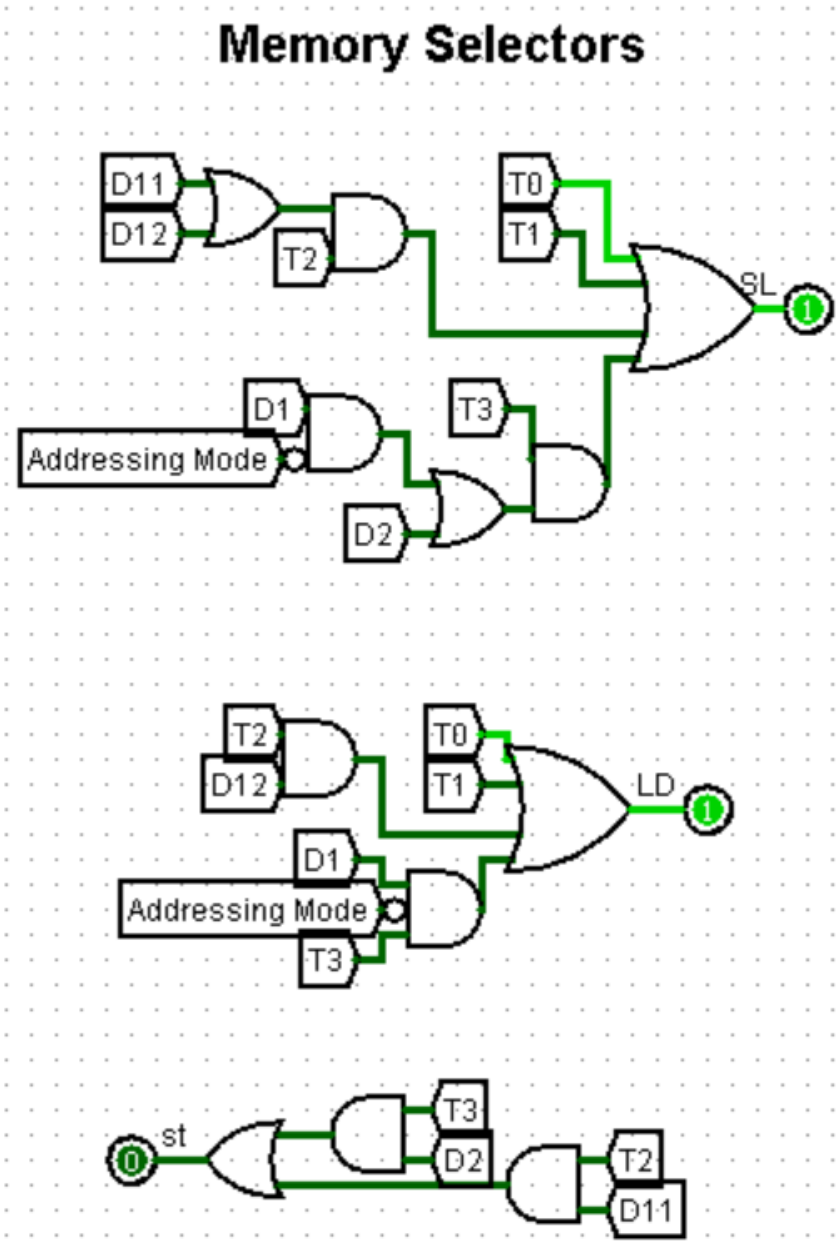


Figure 27: Memory selectors in Control Unit

3.6 MUXes and their selectors

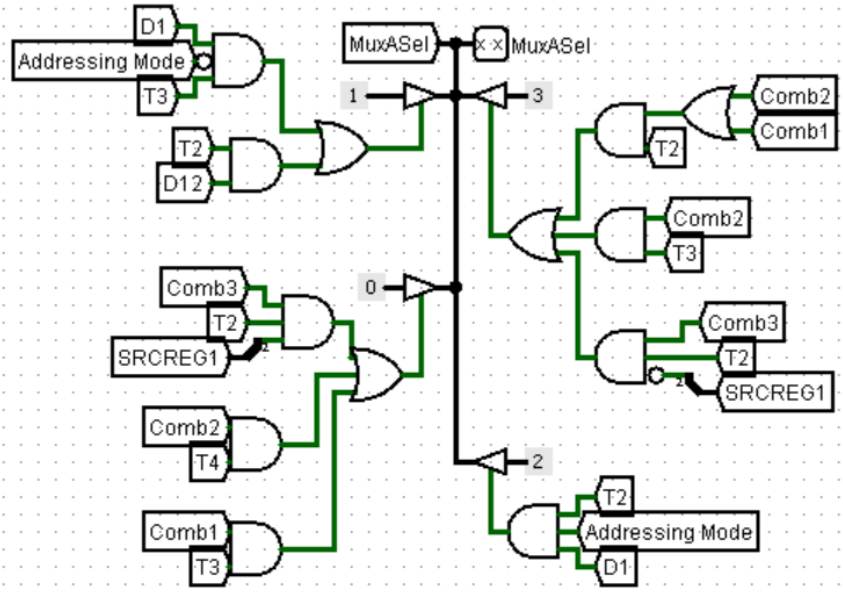


Figure 28: MUX A SEL in Control Unit

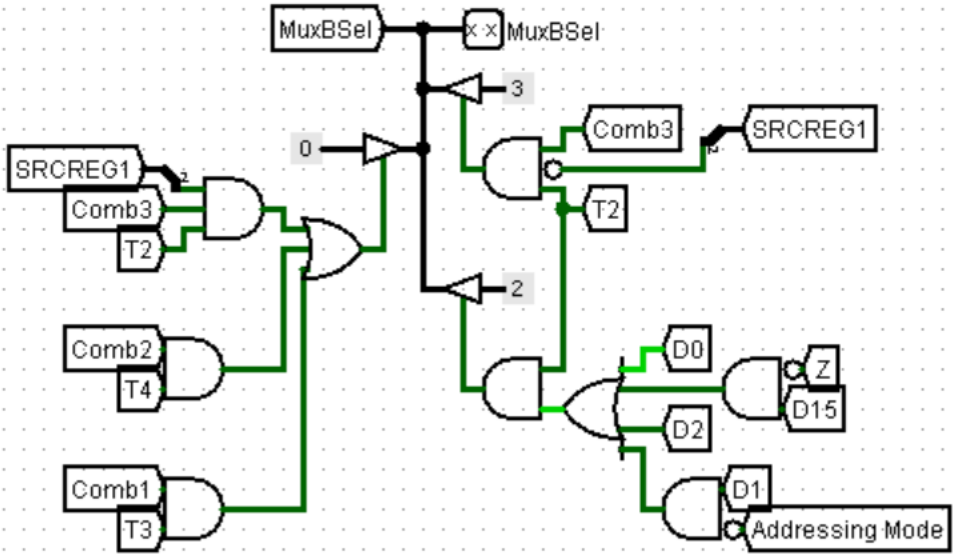


Figure 29: MUX B SEL in Control Unit

3.7 Main Circuit

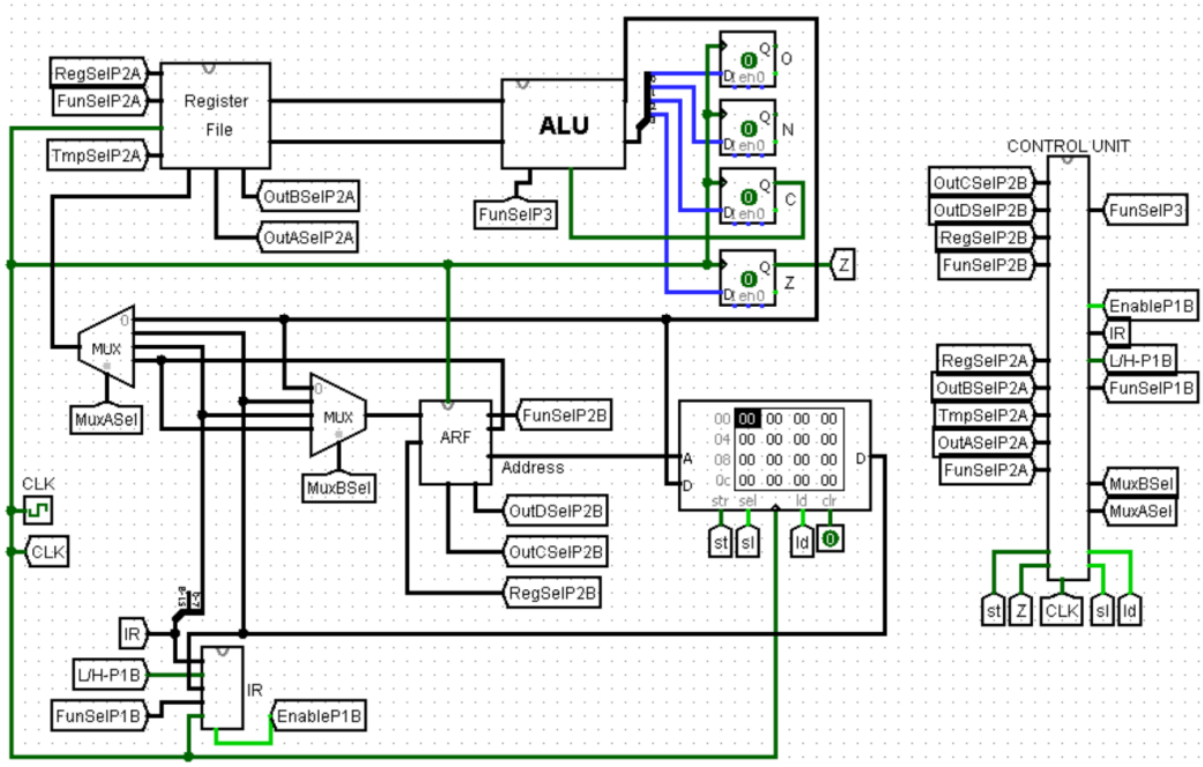


Figure 30: Main Circuit

4 RESULTS

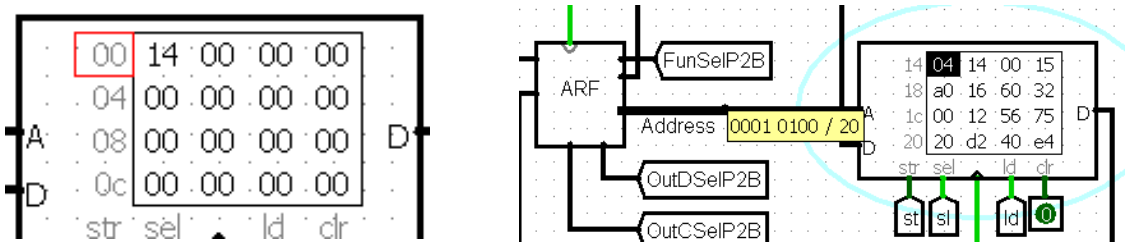
We tried our implementations with the given test cases from the assignment documentation. You can see the steps in below figure 31:

```
        BRA  0x20              # This instruction is written to the memory address 0x00,
                               # The first instruction must be written to the address 0x20


        LD R1 IM 0x05          # This first instruction is written to the address 0x20,
                               # R1 is used for iteration number
        LD R2 IM 0x00          # R2 is used to store total
        LD R3 IM 0xA0
        MOV AR R3              # AR is used to track data address: starts from 0xA0
LABEL:  LD R3 D                # R3 ← M[AR]        (AR = 0xA0 to 0xA4)
        ADD R2 R2 R3           # R2 ← R2 + R3      (Total = Total + M[AR])
        INC AR AR              # AR ← AR + 1       (Next Data)
        DEC R1 R1              # R1 ← R1 – 1       (Decrement Iteration Counter)
        BNE IM LABEL           # Go back to LABEL if Z=0 (Iteration Counter > 0)
        INC AR AR              # AR ← AR + 1       (Total will be written to 0xA6)
        ST  R2 D               # M[AR] ← R2        (Store Total at 0xA6)
```

Figure 31: Summary of our steps, taken from assignment documentation

1. **BRA 0x20:** As seen in Figure 1, BRA opcode means this instruction is written to the memory address 0x00, the first instruction must be written to the address 0x20.



(a) (b) Branching

Figure 32

2. **LD R1 IM 0x04:** As seen in Figure 1, LD means load. We loaded 0x04 to R1. This first instruction is written to the address 0x20, R1 is used for iteration number. But our implementation works a little unexpected, rather than using 0x05 as pdf stated, we had to use 0x04 to make the implementation work properly. So for this part, we changed the test case, but not a significant change.

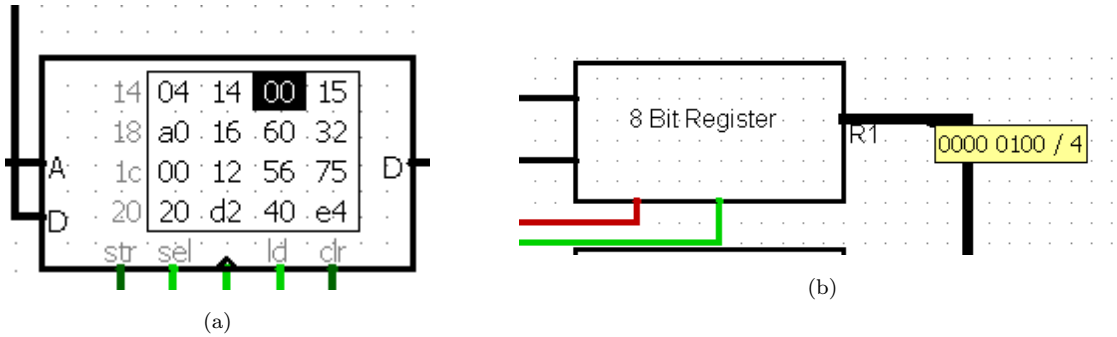


Figure 33: Load R1

3. **LD R2 IM 0x00:** R2 is used to store total

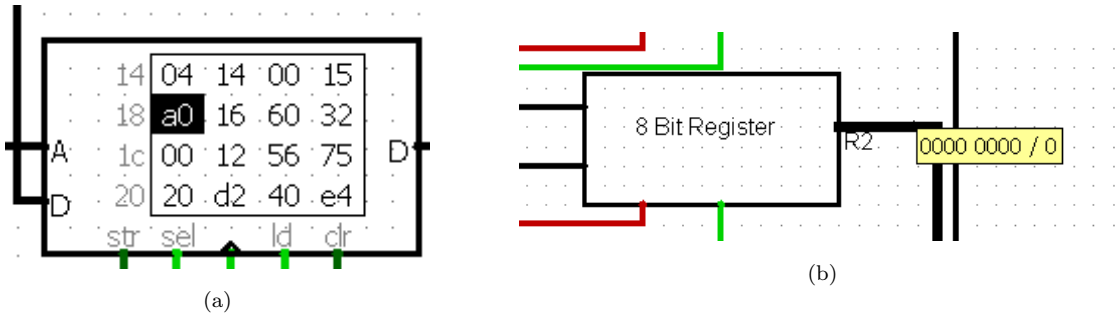


Figure 34: Load R2

4. **LD R3 IM 0xA0:**

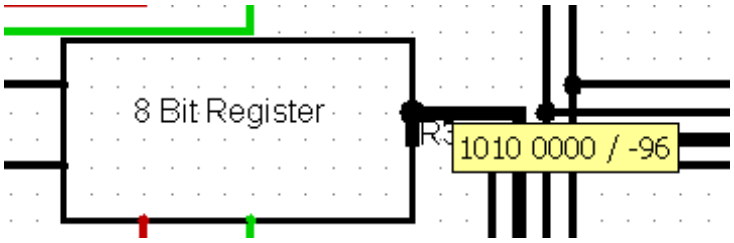


Figure 35: Load R3

5. **MOV AR R3:** AR is used to track data address: starts from 0xA0

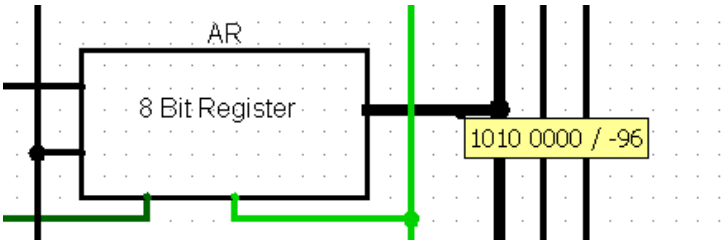


Figure 36: Moving data from R3 to AR

6. **LABEL: (From here, we'll start a loop)** In every loop, the data where AR points is assigned to R3, and then we take the value in R3 and add it to the R2, and written it into the R2 register, where the summation results are hold. After that we increment AR value in order to use it in the other loop, and decrement the iteration value by one. Finally we check if the number of the iteration is equal to zero, we start another loop.

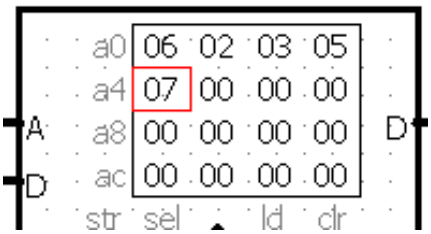
    - LD R3 D: R3 ← M[AR] (AR = 0xA0 to 0xA4). As seen in Figure 1, LD means load. We load R3 directly.
    - ADD R2 R2 R3: R2 ← R2 + R3 (Total = Total + M[AR]). As seen in Figure 1, ADD means addition operation. We took the sum of R2 and R3, and assigned the result to R2.
    - INC AR AR: AR ← AR + 1 (Next Data). As seen in Figure 1, INC means increment. We incremented the value of AR, to use it in the next loop as a pointer to the next location in memory.
    - DEC R1 R1: R1 ← R1 – 1 (Decrement Iteration Counter). As seen in Figure 1, DEC means decrement. We decremented the value of R1 by 1. R1 keeps the total number of loops we should have. So after every loop, we should decrement it to see how many loops we had left.
    - BNE IM LABEL: Go back to LABEL if Z=0 (Iteration Counter ¿ 0)

7. **INC AR AR:** AR ← AR + 1 (Total will be written to 0xA6)

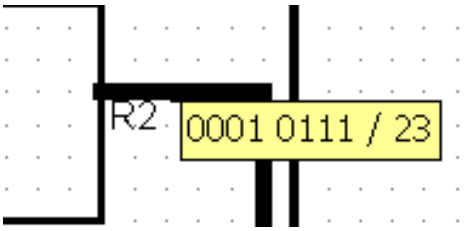
    - In order to leave a space inside the memory, we increment AR by one.

    ST R2 D: M[AR] ← R2 (Store Total at 0xA6)

    - We store the result of the summation inside the 0xA6 address of the memory.



(a) Data to be Read in Memory



(b) Result of the Sum Operation

Figure 37

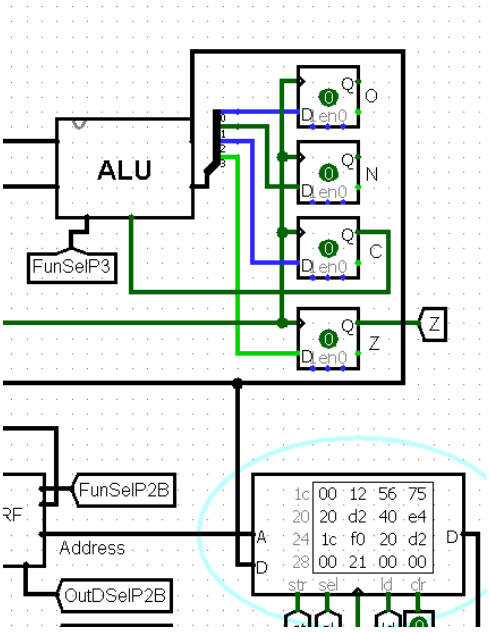
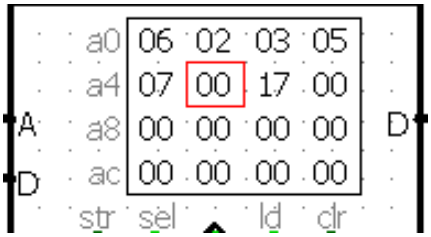
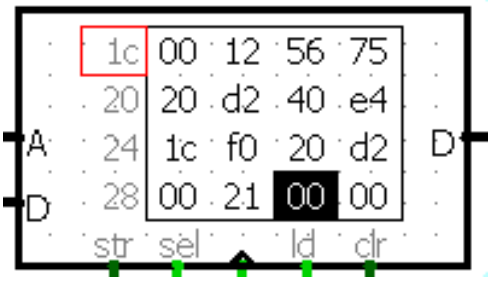


Figure 38: Loading 1 value to Z



(a) Saving the Result of Sum Operation



(b) Memory codes from Label to End

Figure 39

5 DISCUSSION

In this assignment, we are asked to implement to implement a control unit circuit, so that instead of assigning selectors/enables/other inputs manually, we let the circuit decide which operation to do while the given opcodes operate.

We have already implemented the main circuit in the last assignment, so we add a control unit to this main circuit and implement our necessary circuits into this control unit.

At T0 and T1 we FETCH by reading our instructions, which is given in the instruction format from the assignment document, from the memory.

Fetching the operation from memory and decoding it requires 2 clock cycles, at the beginning of every operation. The reason for that is to first fill the LSB's of IR, and then to fill the MSB's of the IR. So during these operations, the FunSel of IR should be on the load mode.

During the first clock signal T0, we will fetch the LSB's of IR, from memory, and load them to IR. For these, we should have the following properties about circuit inputs, which we have already explained in detail back in the corresponding part of the report above.

During the second clock signal T1, we will fetch the MSB's of IR, from memory, and load them to IR. For these, we should have the following properties about circuit inputs, which we have already explained in detail back in the corresponding part of the report above.

At T1, we also DECODE our instruction by getting the Instruction Register's all the bits to the control unit. Then, we splitted them according to our needs. See Figure 2a. After obtaining IR's bits, we picked the OPCODE bits from it and decoded the opcode bits into tunnels, which we will later use in out implementations as inputs specifying the operation. See Figure 2b for visual explanation, and we're done fetching and decoding.

For D0, after after fetching at T0 and decoding at T1, during the operation 0x00, we are trying to assign the value to PC. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to addressing modes given in the pdf, the addressing mode is immediate, so the Value stands for ADDRESS Field. So we will load the value in the ADDRESS Field to PC.

Since we already have the ADDRESS Field loaded to IR(during T0 AND T1), the only thing we need to do is to set MuxBSel in a way to make ARF access the ADDRESS Field. To do that, we will assign the MuxBSel to 10. And then, we should enable the PC to fetch the data came from ADDRESS Field via Mux B.

For D1, after fetching at T0 and decoding at T1, during the operation 0x01, we tried to assign the value to Rx. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to addressing modes given in the pdf, we have two different addressing modes. We need to consider both of them.

If addressing mode is DIRECT, we will assign the value in M[AR] to Rx. To do that, we will do the steps which we have already explained in the corresponding part of the report above.

If addressing mode is IMMEDIATE, we will assign the value in ADDRESS Field to Rx. To do that, we will do the steps which we have already explained in the corresponding part of the report above.

For D2, after fetching at T0 and decoding at T1,during the operation 0x02, we are trying to assign the Rx to Value. While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to addressing modes given in the pdf, the addressing mode is direct, so the Value stands for M[AR]. So we will load the value in the Rx to M[AR]. To do that, we will do the steps which we have already explained in the corresponding part of the report above.

For D3-D13-D14, after fetching at T0 and decoding at T1, during the operation 0x03, we are trying to assign the SRCREG1 to DESTREG. And during the operations 0x0D & 0x0E, we are assigning the SRCREG1 to DESTREG, too. But just with a little additional clock cycle, which will activate the FunSel of DESTREG, so we first will assign the data to DESTREG and then either increment or decrement it in its place. That's why we combined these operations.

While doing that, we should first decide whether the DESTREG and/or the SRCREG1 is on the ARF or Register File, to find out what the path data will follow is. According to the DESTREG/SRCREG1/SRCREG2 table from pdf, there are 4 different cases that changes the data path. These data paths and what to do when each one of them is encountered, are explained in detail one by one in the corresponding part of the report above.

From these opcodes, we have already managed to transfer SRCREG1 to DESTREG. All we have to do left, is to decide where DESTREG is (whether the ARF or Register File) and then give one more clock cycle with arranged FunSel to, decrement operation for D14 and increment operation for D13. After that we made some other arrangements. For that, we will follow some steps to arrange funsel of correct part, so we will follow the steps explained in detail one by one in the corresponding part of the report above.

For D6-D9-D10, we grouped them into Comb1. In these operations, first we fetch at T0 and decode at T1.

Then at the T2, we take corresponding value from ARF, which is indicated with last2 bit of SRCREG1, such as for 10 we take AR regardless and assign it to TempReg3 ofmost significant 2 bits. We assign this value to corresponding temporary register.

After that, at the T3, we decide if the value is from ARF or Registers. According to this infowe give the proper value to ALU, make the neccessary operations and assign the outputof ALU to DESTREG.

For D4-D5-D7-D8, we grouped them into Comb2. In these operations, first we fetch at T0 and decode at T1.

At the T2 and T3, we take the corresponding value from ARF according to last 2 bit of SRCREG1/SRCREG2, such as for 01 we take PC regardless of most significant 2 bits. And assign this value to Temporary Register.

Then at T4, we decide whether the value is coming from ARF or Register, and if both of the values are coming from ARF then we take two corresponding Temporary Registers' values, if only one is coming from ARF we take one corresponding Temporary Register's and one corresponding Register's value, if none of them is from ARF then we take two of the corresponding Registers' values and give them to the ALU in order to make the necessary operations and assign the output of ALU to DESTREG.

For D11, we did not grouped D11 into a combination, because it would not be useful. As always in these operations, first we fetch and decode at T0 and T1.

And at the T2, firstly we take the value from the corresponding register of Register File and also take the value of SP from ARF. And then we give these values to the memory to store the data in the appropriate location in the memory and also set store and select inputs of memory to true.

And at the T3, we decrement the SP value in the ARF by giving the appropriate inputs given below to ARF and make SP to decrement itself.

For D12, we did not grouped D12 into a combination, because it would not be useful. As always in these operations, first we fetch and decode at T0 and T1.

And at the T2, firstly we need to increment the SP value inside the ARF according to the table's 0x0C line. We increment the SP value by giving ARF to appropriate input

values and make the SP register to increment itself.

After that at the T3,we take the value of SP from ARF and also take the value from the corresponding address SP of the memory. And then we give these values to the Register File to load its register via MultiplexerA. We set this multiplexers input to give the data to register file in order to load this data from the SP address of the memory to Register File's corresponding register.

For D15, after fetching at T0 and decoding at T1, during the operation 0x0F, we are trying to assign the value to PC, but in addition to that, we have a requirement: Z have to be 0.

While doing that, we should first decide the addressing mode, to find out what the "VALUE" is. According to addressing modes table from pdf, the addressing mode is immediate, so the Value stands for ADDRESS Field. So we will load the value in the ADDRESS Field to PC, if Z == 0. So we can do the exact same operations made during 0x00, just with an additional Z requirement.

Since we already have the ADDRESS Field loaded to IR(during T0 AND T1), the only thing we need to do is to set MuxBSel in a way to make ARF access the ADDRESS Field. To do that, we will assign the MuxBSel to 10, if Z==0. Then, we should enable the PC to fetch the data came from ADDRESS Field via Mux B.

6 CONCLUSION

In conclusion, in this assignment we implemented a whole control unit which operates according to the criterias given in the assignment document. It was really hard to figure out how to construct our control unit so that it yields the necessary inputs and their logics to the main circuit. Since we are already experienced from Digital Circuits Course and Computer Organization Course, after struggling for a while we succeed to implement the circuit. In this assignment again, We practiced creating circuits and implementing them using Logisim. Again we have seen that Logisim is very useful for implementing the circuits. We were able to get diagrams and outputs of the circuits easily, in Logisim.

For some opcodes, for the sake of the simplicity of the logic and complexity of the circuits, we grouped them into some combinations. But for those, which would not be easier to group, we implement their circuits as they are. Even in the same combinations, there was still some differences in opcodes operations, so we also implemented our control unit according to this differences.

We did test our whole circuit with the test case given in the assignment document and verified that we conclude our work successfully.

After all, we practiced our knowledge on the LaTex language and Logisim program. This experiment was harder than the previous one but as we are getting more comfortable while using programs and languages and working as a team, we concluded our work successfully.