ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 2

DUE DATE : 09.06.2021

GROUP NO : G23

GROUP MEMBERS:

150180010 : ELİF ARIKAN

150180112 : ÖMER MALİK KALEMBAŞI

150180052 : HÜSEYİN AVERBEK

1 INTRODUCTION

In this project, we created a control unit for our ALU and register system from the first project. This control unit performs the 19 actions described in the project and has access to four numbered registers R0, R1, R2, R3 as well as registers SP, AR, PC. We also employ a 16-bit instruction register IR.

2 PROJECT PARTS

2.1 SEQUENTIAL COUNTER

This is an 8-bit register that clears itself when the reset value is 1, and rises when the reset value is 0. Its output is connected to an 8-bit decoder, and the outputs of the decoder are sent through tunnels T0 to T7.

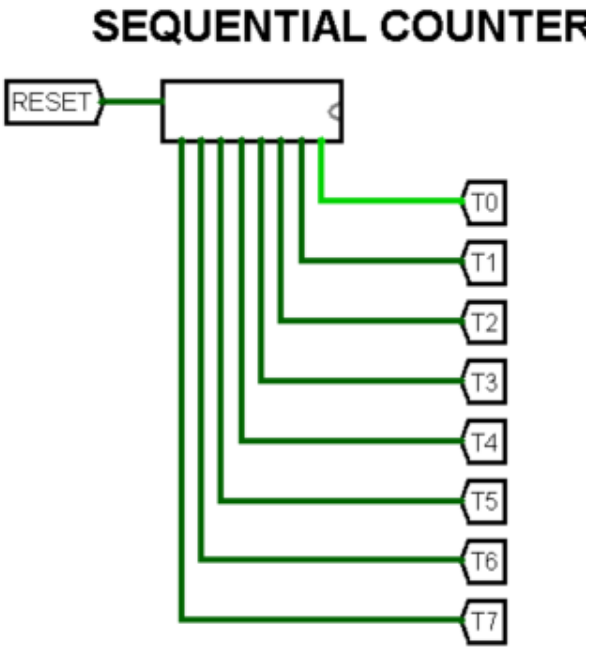

Figure 1: Logisim Circuit For Sequential Counter

2.2 Fetch Cycle

Two timing signals are received by our fetch cycle. In T0, we use the address on the PC to navigate to the memory location, copy it to IR(15-8) and raise the PC. In T1, we perform the same thing, except we copy the value to IR (7-0). Following this cycle, we begin performing our operations.

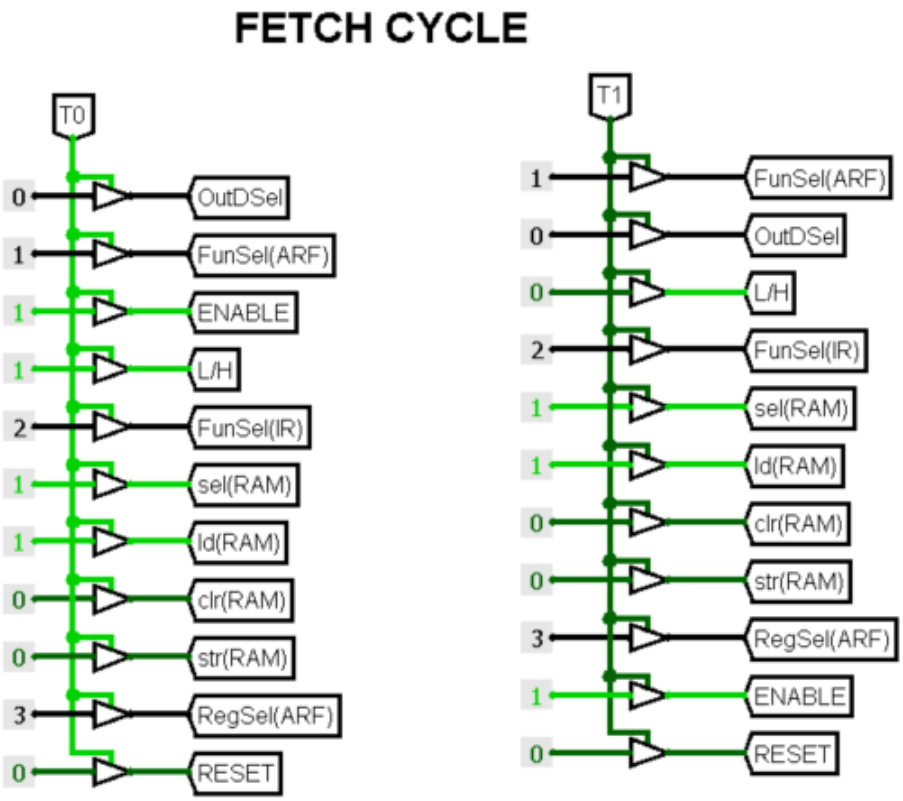

Figure 2: Logisim Circuit For Fetch Cycle

2.3 DECODE

In decode operation, we designed a 4:16 Decoder to decode first MSB 4-bit of the Instruction which is OPCODE. ENABLEdecode is activated with T2. While doing this, we closed all registers located in ARF and RF with using their RegSel's.

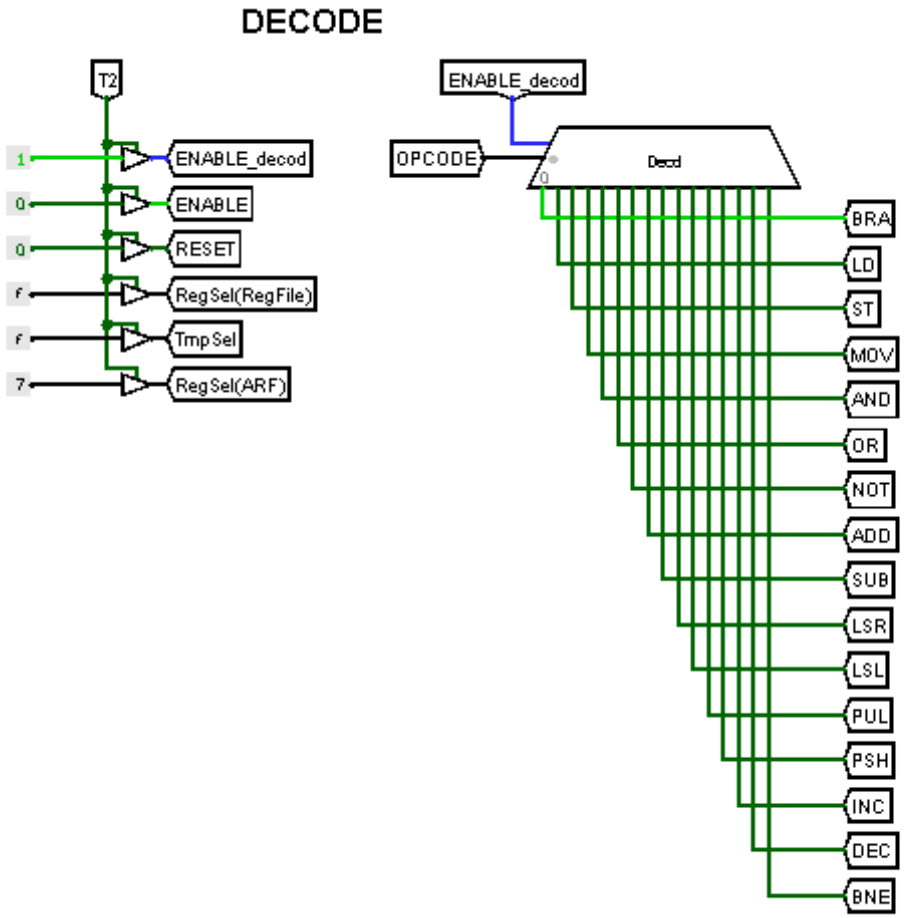

Figure 3: Decode Process

2.4 AUXILIARY CIRCUITS WE USED IN THE PROJECT

In this project, we designed small auxiliary circuits to save us time and provide convenience. First of them is a multiplexer, takes first two bits of DESTREG and gives proper RegSel(ARF) which is ARFRegSel.

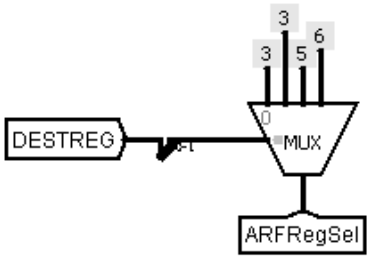

Figure 4: Multiplexer for ARFRegSel

Secondly, we designed small decoder to convert REGSEL to desired Register Rx.

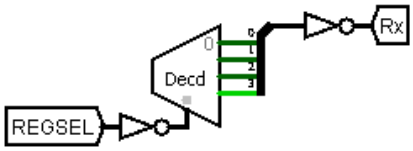

Figure 5: Decoder for Rx

We designed another decoder to convert DESTREG to desired RegSel(RegFile) which is REGFILE RegisterSel.

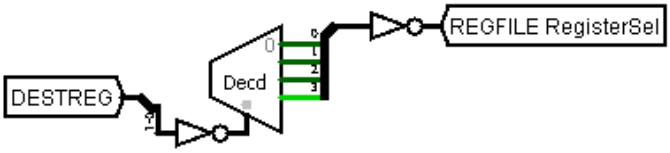

Figure 6: Decoder for REGFILE RegisterSel

We designed multiplexer to get proper FunSel(ALU)'s for related operations. Multiplexers select input is 4-bit OPCODE.

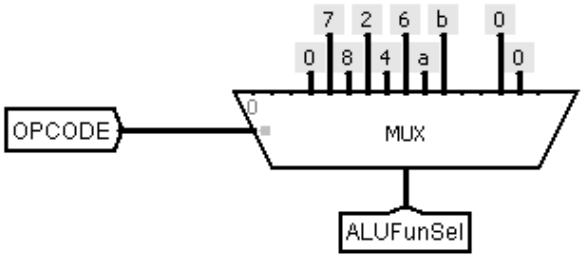

Figure 7: Multiplexer for ALUFunSel

We designed small multiplexer for converting OPCODE to desired FunSel for INC and DEC operations.

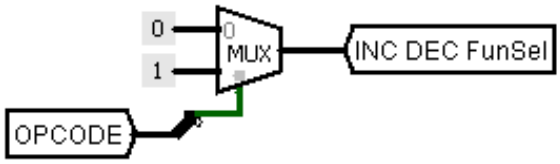

Figure 8: Multiplexer for INC DEC FunSel

2.5 BRA 0X00 and BNE 0X0F

This operation is carried out with the use of two time signals. These are address-referenced operations, but they only support immediate addressing, hence bit 8 of the instructions is set to 0. We don't need to access memory for these reasons. In this method, we use MuxBSel to write the first 8 bits of the command to the PC in part2b from first project.

If the Zero flag in hw2part2.circ is set to zero, the BNE operation is performed. The remainder of the BNE operation is the same as the BRA operation (PC Value) after checking the zero flag.

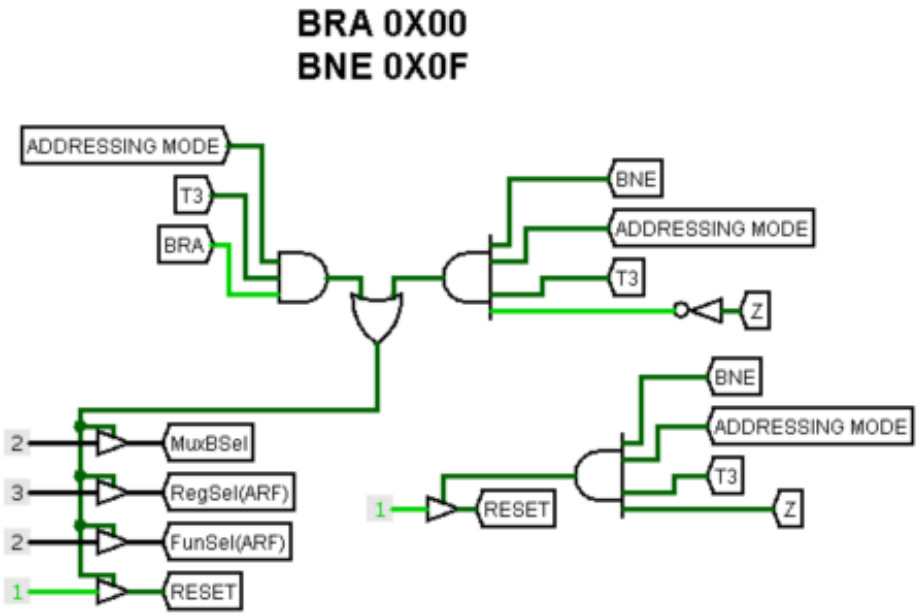

Figure 9: Logisim Circuit For BRA and BNE Operations

2.6 LD 0x01

In this operation if the addressing mode (8th bit) is 1 we directly write the value, that is the first 8 bits, into the desired register which was given in the instructions bits 9-10. If the addressing mode is 0 then we need to access to the memory location of the address register and write the value at that location of the memory into the desired register.

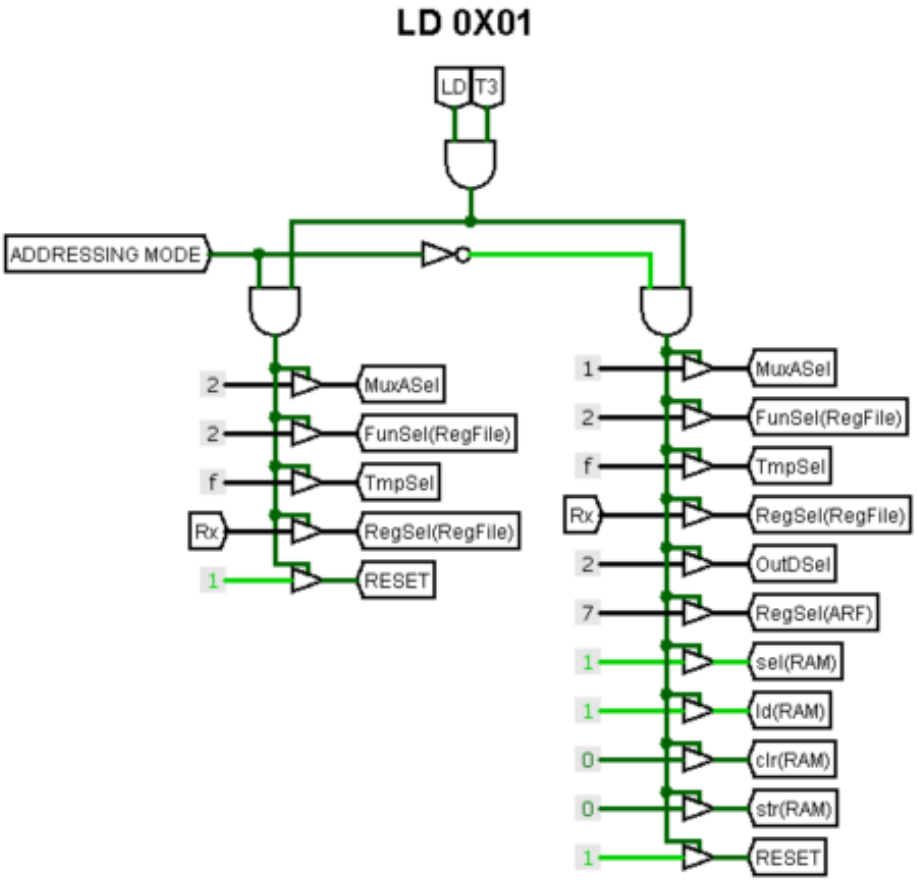

Figure 10: Logisim Circuit For LD Operation

2.7 ST 0x02

In this operation, we must write the desired register value provided in REGSEL to AR's memory location. Bit 8 of the instruction is set to 0 since this operation is address-referenced but only permits direct addressing.

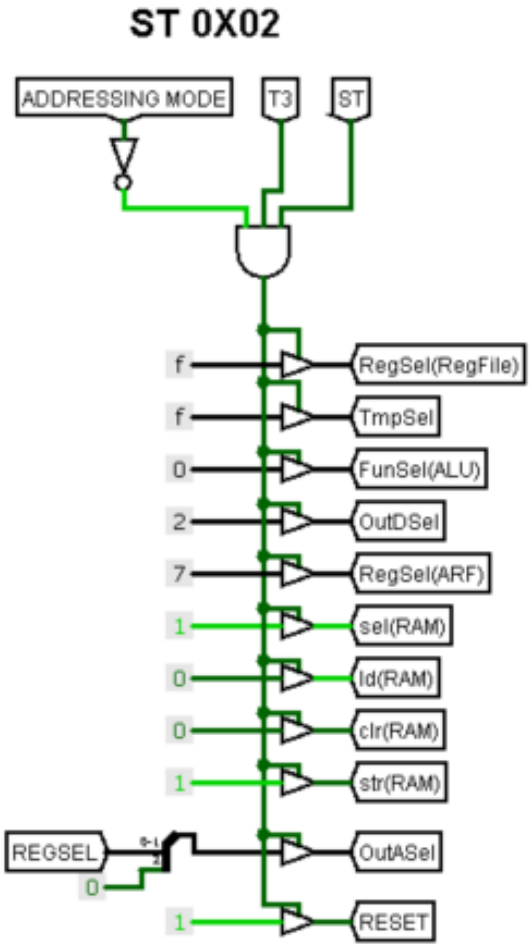

Figure 11: Logisim Circuit For ST Operation

2.8 PSH 0x0C and PUL 0x0B

These operations function similarly to real-world stack operations on data structures. In the PSH operation, we first write the value to the memory address of SP (stack pointer) in ARF from first project, and then we decrement the value of SP. For PSH operation, one clock signal is sufficient to sending value in M[SP] and decrementing SP.

The PUL operation is similar to a stack operation. In this operation, we increase the value in SP, then locate the slot of memory SP points and write the value there to the memory address containing the desired register points Rx M[SP]. Here one clock signal is not enough. We firstly increment the SP value in ARF, in second clock cycle (T4) we take the value in changed M[SP].

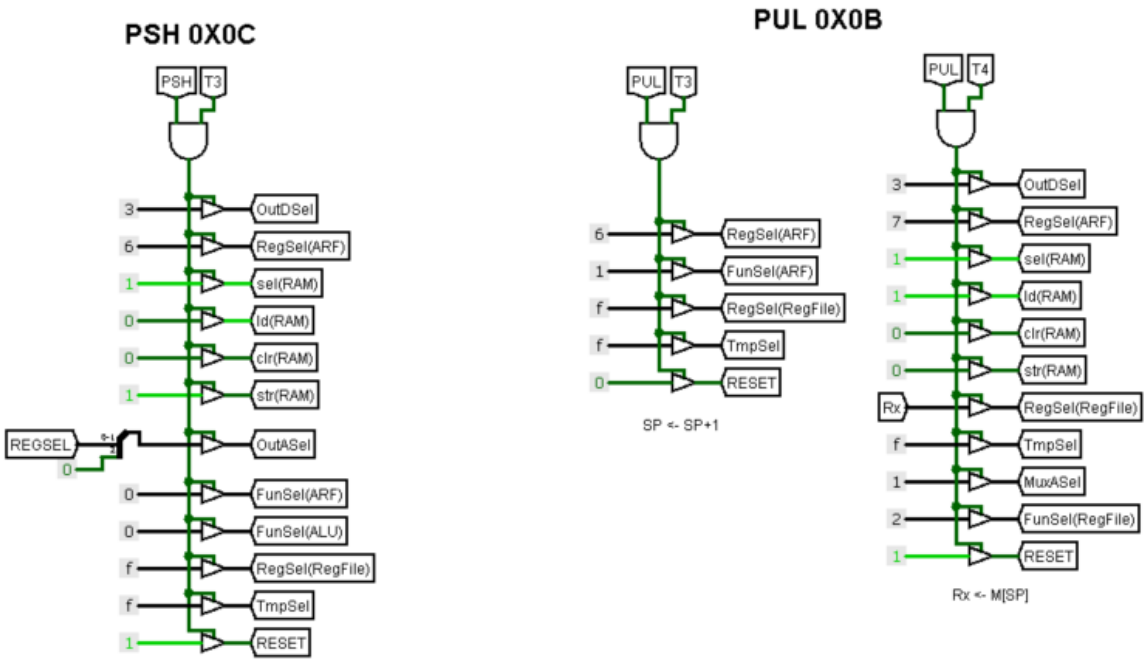

Figure 12: Logisim Circuit For PSH and PUL Operations

2.9 MOV 0x03, NOT 0x06, LSL 0x09, LSR 0x0A

These four operations are basically similar to each other because we only used SR-CREG1 and DESTREG. As a function, they pass the values they get from the source register to the desired operations and save them to the destination register. Then, we implemented a circuit that checks which units' register is our source and which units' register is our destination. Here we have four different cases: RF to RF, RF to ARF, ARF to ARF and ARF to RF, sources and destinations, respectively. All the cases we used a ALU FunSel tunnel (ALUFunSel) which we designed earlier. One clock cycle was enough to apply RF to RF and RF to ARF. Thus, these cases will be completed after T3 ends.

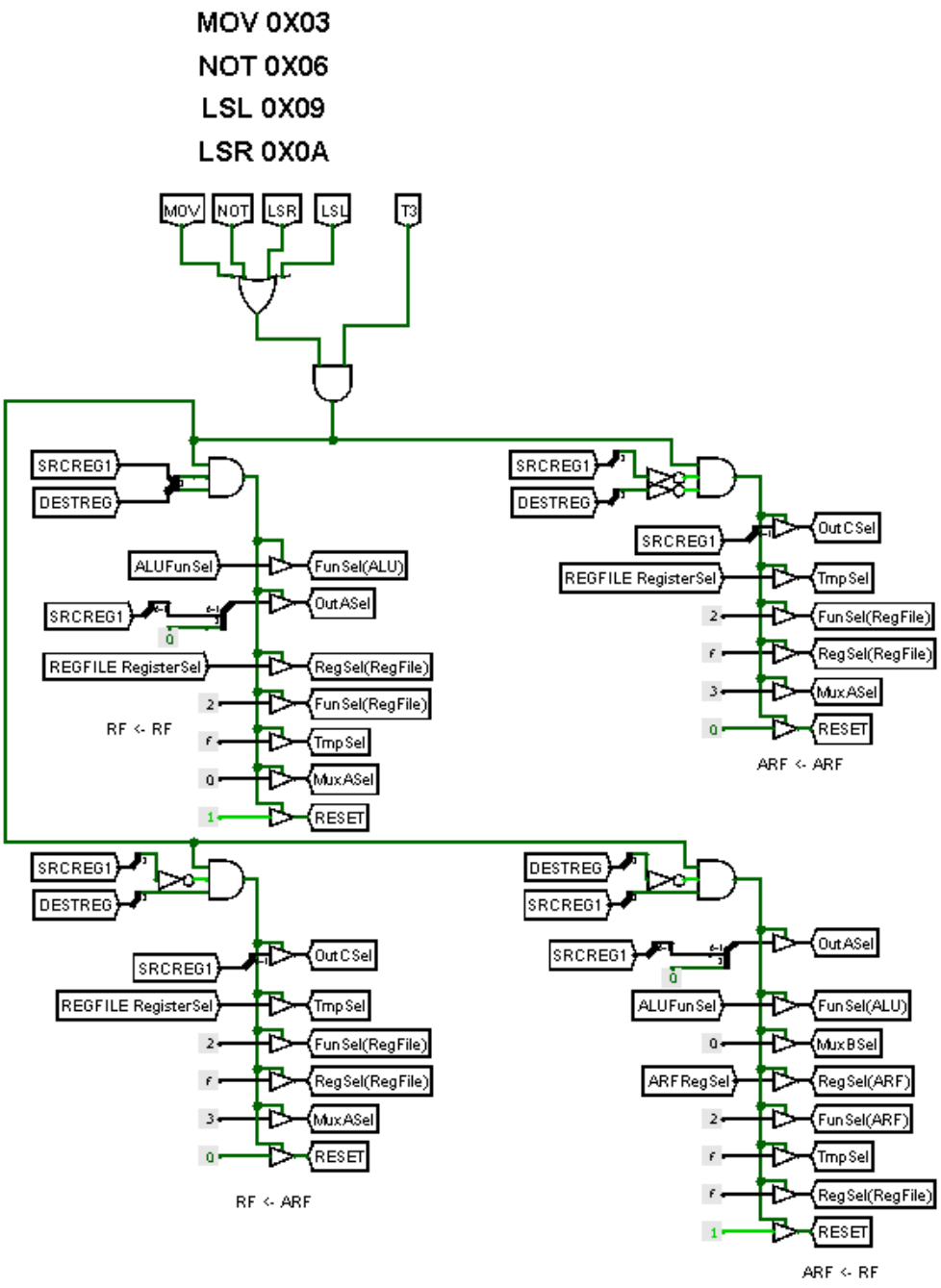

Figure 13: Logisim Circuit For MOV, NOT, LSL andLSR (T3)

On the other hand, for the cases ARF to RF and ARF to ARF, one clock cycle was not sufficient. So, we firstly took the register value which is located in ARF and sended it to RF through MuxA. Here we closed all general-purpose registers by equating RegSel(RegFile) to 0xf and we opened a temporary register to store the value. Thus, T3 is completed. In next clock cycle T4, OutASel selects the temporary register which stores the value and sends the value to ALU. Connected tunnel ALUFunSel selects ALU operation to implement. According to destination, OutALU passes MuxA or MuxB and

arrives the RF or ARF. Then T4 is completed.

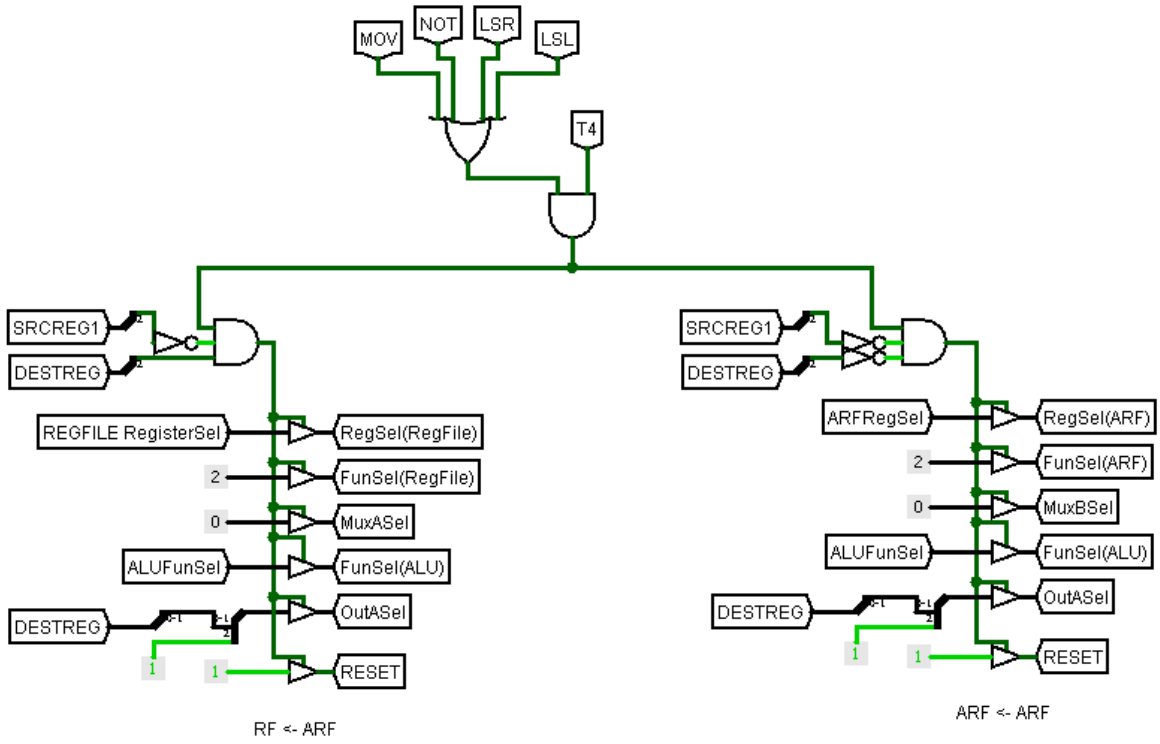

Figure 14: Logisim Circuit For MOV, NOT, LSL and LSR (T4)

2.10 INC 0x0D, DEC 0x0E

These two operations are similar because we changed only FunSel(RegFile) or FunSel(ARF). We used FunSel's for incrementation or decrementation. In these operations, our clock cycles goes to T6. In first clock cycle T3, we designed a circuit to move the value of source register (SRCREG1) to destination (DESTREG). There are four different cases: RF to RF, RF to ARF, ARF to RF and ARF to ARF. One clock cycle sufficient for implement this. We used REGFILE RegisterSel tunnel here, which we designed earlier to select RegSel of RF. All FunSel's are set as 2-bit 10 (LOAD). OutASel or OutCSel comes from SRCREG1 from Instruction. MuxA and MuxB selects destinations.

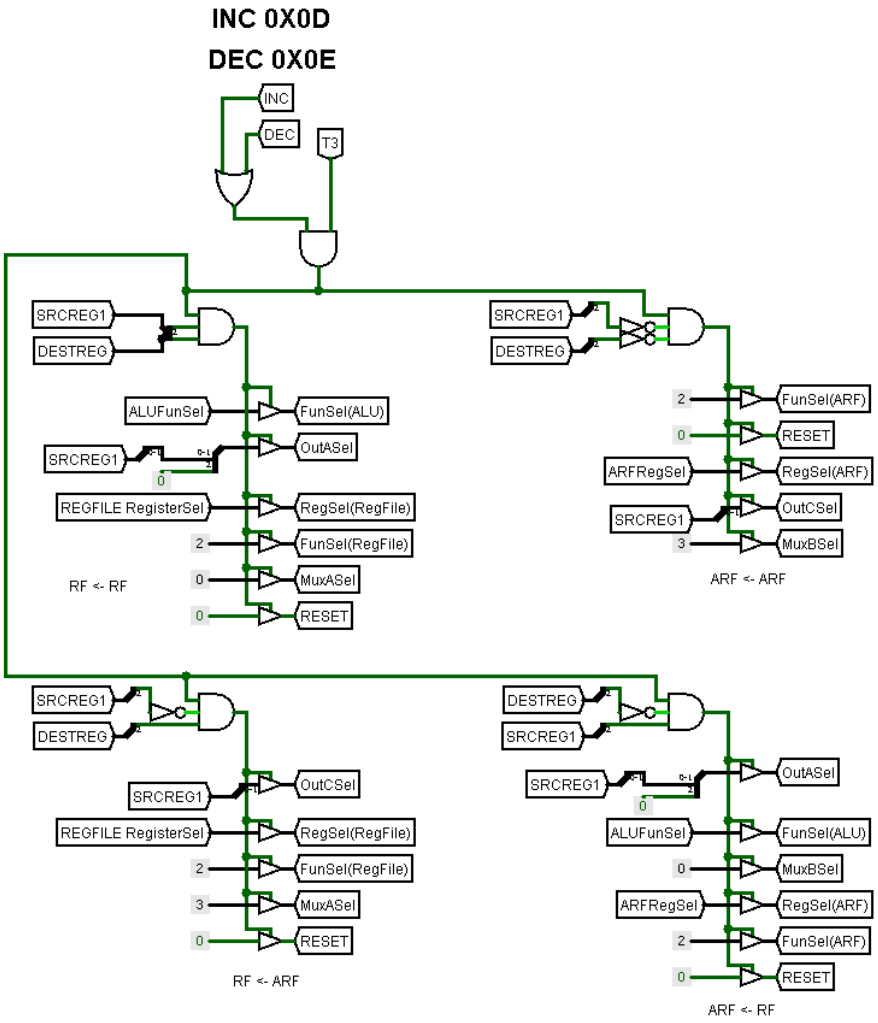

Figure 15: Logisim Circuit For INC and DEC (T3)

In next clock cycle T4, RF or ARF implements increment or decrement operations in itselves. In ARF, RegSel(ARF) connected to tunnel ARFRegsel which we designed earlier, in RF, RegSel(RegFile) connected to tunnel REGFILE RegisterSel which we designed earlier. ARF and RF FunSel's are connected to tunnel INC DEC FunSel which we designed for these operations. After this clock cycle, increment or decrement orders will be done.

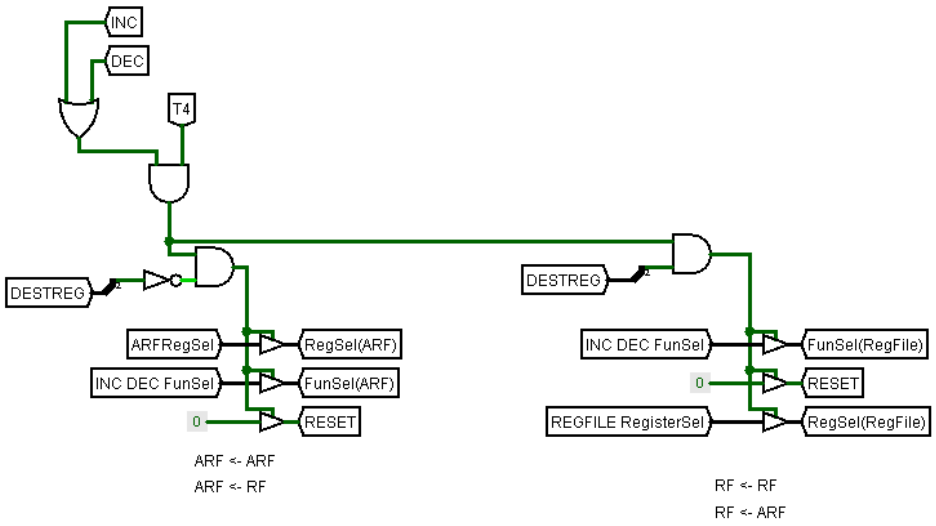

Figure 16: Logisim Circuit For INC and DEC(T4)

In clock cycle T5, we started to save the results to ZCNO Flag Register located near the ALU. We need to pass the value, inside the destination register, through ALU to do the implementing operation correctly. In RF to RF and ARF to RF operations, T5 cycle is sufficient to to this. We get the value through OutASel, here FunSel(ALU) comes from the tunnel ALUFunSel which we designed earlier. After the value passes through the ALU, ZCNO flag register will be activated and worked. On the other hand, in operations ARF to ARF and RF to RF, one clock cycle will not be enough. Here we followed a different way, firstly we took the value from ARF and stored in related temporary register in RF. We closed all general-purpose registers for this. FunSel of RF set to 2-bit 10 (LOAD). After this implementation, value is stored in a temporary register and T5 cycle is end.

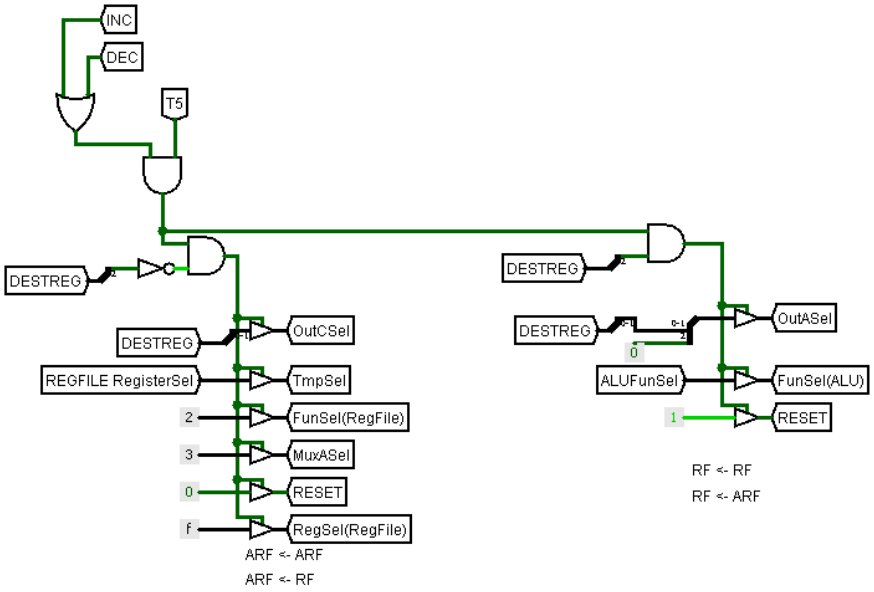

Figure 17: Logisim Circuit For INC and DEC(T5)

The clock cycle T6 only works for the operations ARF to ARF and RF to ARF. Firstly we get the value which is recently stored in temporary register with OutASel. Then the value passes through ALU and activated ZCNO Flag Register. Here FunSel(ALU) connected to tunnel ALUFunSel which we designed earlier. To complete operation correctly, we set RESET tunnel to 1.

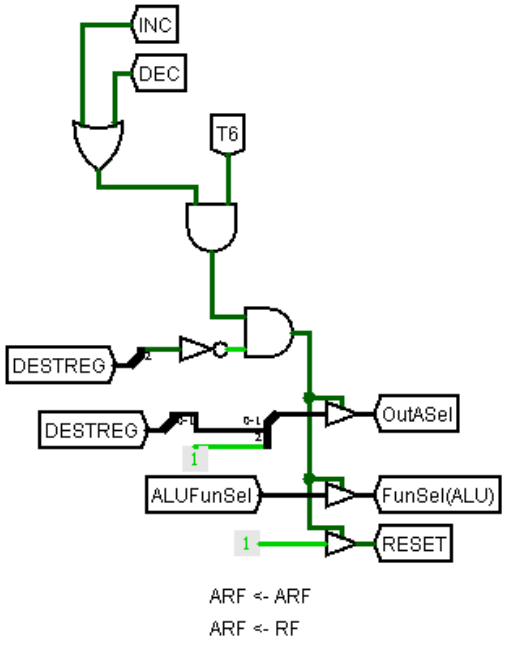

Figure 18: Logisim Circuit For INC and DEC(T6)

2.11 AND 0x04, OR 0x05, ADD 0x07, SUB 0x08

First of all, we designed four different operations together in this part, because what is required from us in these operations is to perform the desired operations on two different source registers and load them into the destination register.As stated in the homework file, we can determine which type of register it is according to the binary code contained in the 4-bit registers. As we can see, since the most significant bits of all registers are 0, we decide whether we will operate from ARF or RF by looking at the second bits of the registers.If the second bit of the register is 0 then the operation is done via ARF, otherwise via RF. We will operate with three different registers: DESTREG, SRCREG1, SRCREG2. As I explained above, there are 2 different states for each of them, ARF or RF. Therefore, we are managing these operations for 8 different states from the 2^3 state. Now, Let's explain this 8 different states.

1. RF ← RF RF

   In this process, we took the A and B outputs of the SRCREG1 and SRCREG2 register file and sent them to ALU within a single clock signal, we connected the ALUFunSel tunnel, which we arranged according to the desired operation, to the funsel input of Alu according to the type of operations (addition, substruction, and, or). By the way, We loaded the value we got from the output of ALU with MuxASel to the Register File according to the DESTREG value.

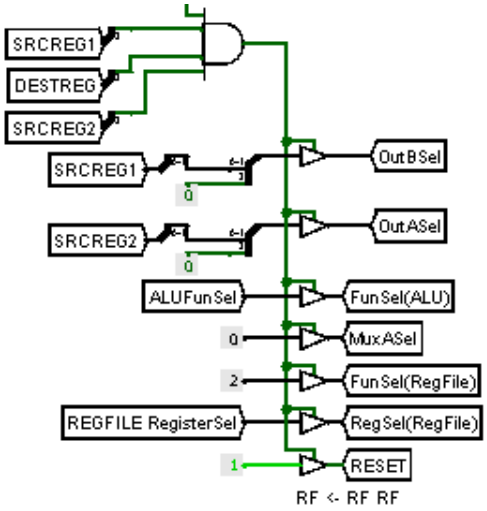

Figure 19: RF ← RF RF operation in clock T3

2. RF ← ARF RF

   In this case, SRCREG1 is a register kept in the ARF. In the first clock, we took the register specified by SRCREG1 from the outC output of ARF and loaded a temp

register kept in the register file. In the next clock, taking the value we have assigned to the temp register from the A output of the register file, we got SRCREG2 from output B. We passed these values from the ALU according to the desired operations and loaded the output into the register file according to DESTREG.

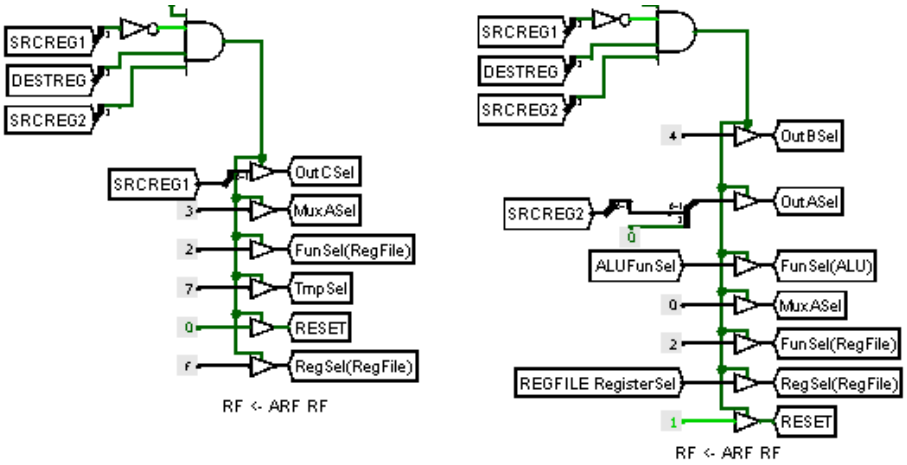

Figure 20: RF ← ARF RF operation in clock T3 and T4

3. RF ← RF ARF

   In this case, SRCREG2 is a register kept in the ARF. In the first clock, we took the register specified by SRCREG2 from the outC output of ARF and loaded a temp register kept in the register file. In the next clock we did the same as we did in case 1. Since our 2 source registers are in the register file, the continuation of the process proceeds in the logic of the RF ← RF RF operation.

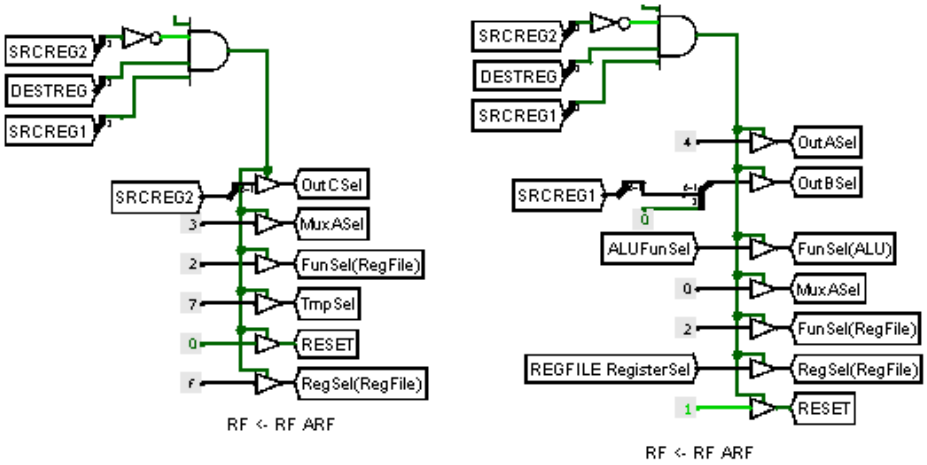

Figure 21: RF ← RF ARF operation in clock T3 and T4

4. RF ← ARF ARF

Both of our source registers in this process are from the registers kept in ARF. In this case, we assign SRCREG2 to one of the temp registers at the first clock. In the second clock, we assign SRCREG1 to another temp register. At the last hour, since our 2 source registers are now in the register file, we designed the desired operations by doing the same thing we did in the RF ← RF operation and loaded them into the register given as DESTREG.

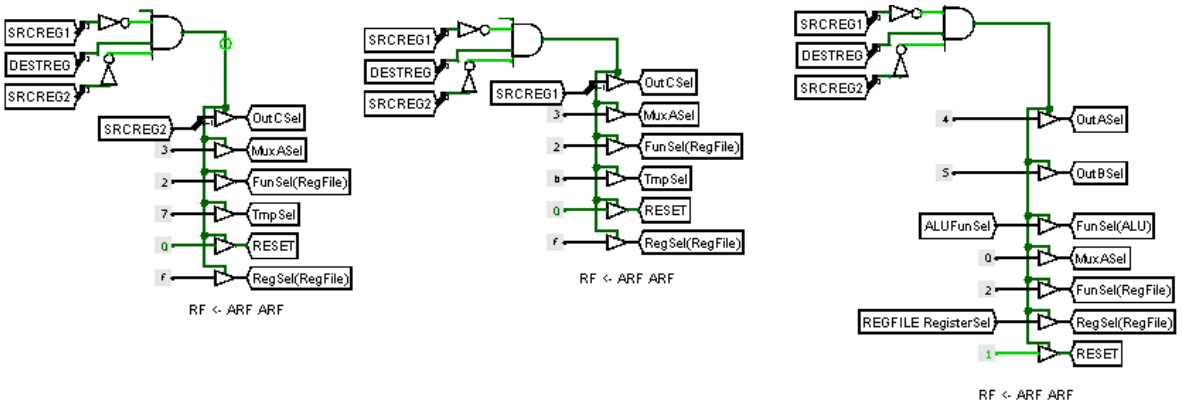

Figure 22: RF ← ARF ARF operation in clock T3, T4 and T5

5. ARF ← RF RF

   In this process, we took our source registers from the register file and passed them through the ALU, similar to the 1st case(RF ← RF RF), and uploaded them to the Address Register File. This time, we gave the value we got from the output of the ALU using MuxBSel as input to the ARF. At the same time, we set the value of the FunSel of ARF to load as 10, and arranged the RegSel of ARF according to DESTREG.

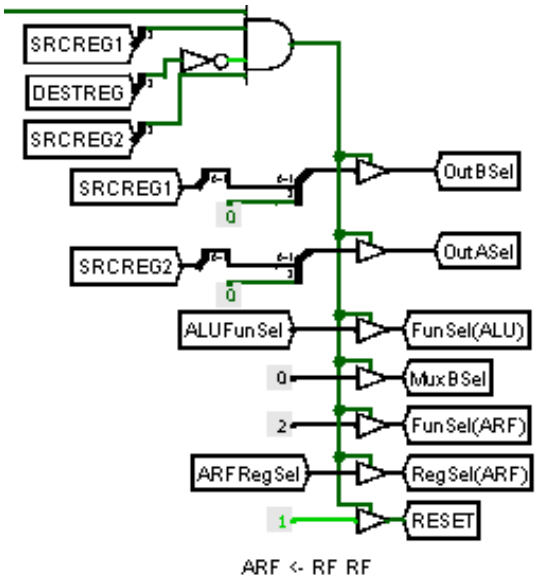

Figure 23: ARF ← RF RF operation in clock T3

6. ARF ← ARF RF

   First of all, we took SRCREG1 kept in ARF with OutCSel output of ARF and gave it as input to REgister File via MuxASel and uploaded it to a Temp Register. In the next clock, we took the temp register that we loaded with the OUtBsel of the Register File, and got the SRCREG2 with the OutASel. We have given them to the ALU as inputs. Afterwards, we gave the process output from the ALU as input to ARF again via MuxBSel and loaded it into DESTREG.

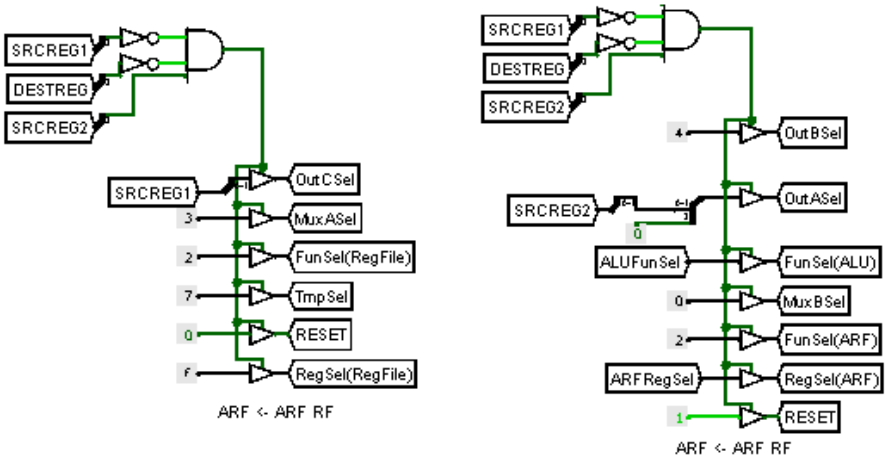

Figure 24: ARF ← ARF RF operation in clock T3 and T4

7. ARF ← RF ARF

   In this part, we pulled SRCREG2 from ARF and loaded it into a Temp Register

kept in Register file. The next thing we did in the clock is ARF ← RF RF operation.

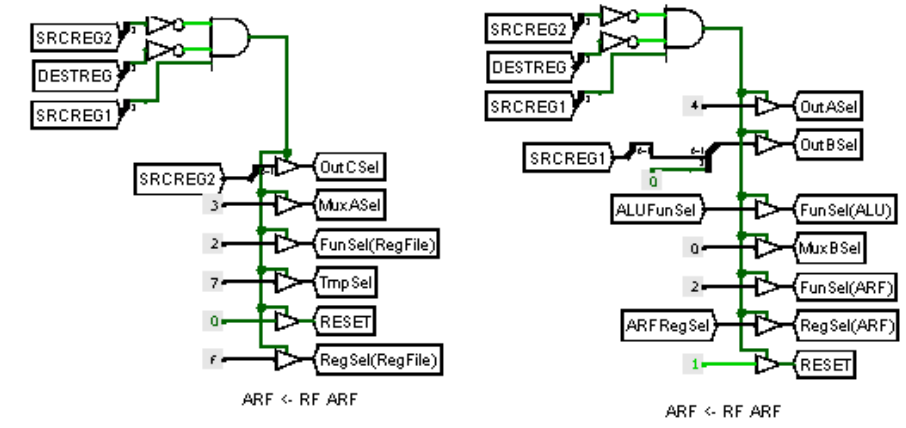

Figure 25: ARF ← RF ARF operation in clock T3 and T4

8. ARF ← ARF ARF

   What we do in this process is essentially similar to what we did in cases 6 and 7. In the first clock, we loaded SRCREG2 into the temp register. At the second clock, we loaded SRCREG1 into another temp register in Register File. At the third clock, both of our source registers are now kept in the register file as temp registers. We took our source registers that we uploaded to Temp Registers from the outA and outB outputs of the Register File and gave them as input to the ALU. We performed the desired operations in the ALU using the ALUFunSel tunnel. And we took the output of the ALU and loaded onto DESTREG in the ARF.

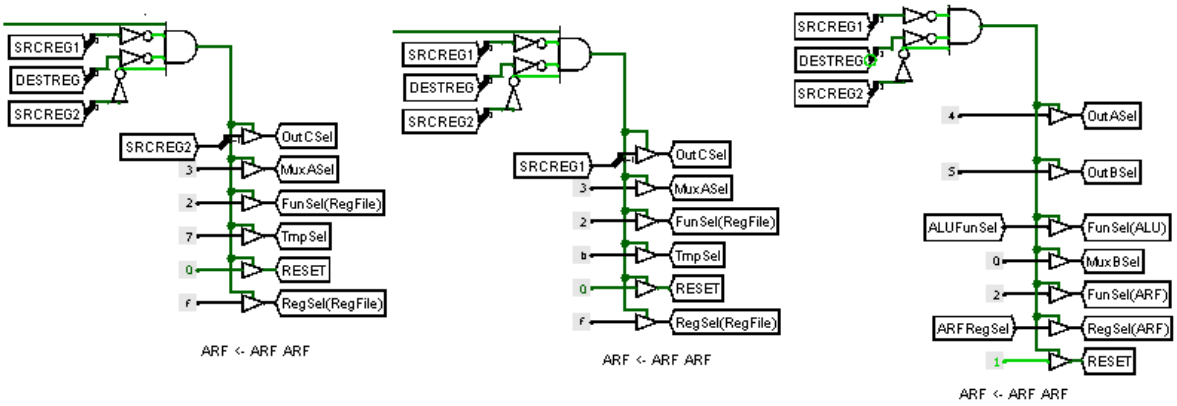

Figure 26: ARF ← ARF ARF operation in clock T3, T4 and T5

3 RESULTS

In the example given, we were asked to total of the values held at 5 consecutive addresses of the memory and write them to desired address in the memory. // We prepared a text file according to the commands written in the assignment. For the BRA 0x20 transaction, we wrote it as IR 0420. We continued to write from 0x20th address for the following transactions. For LD operations, we set the IR to be 1405,1500,16a0 in order. The command of IR for MOV operation is 3260 . Then, the commands in LABEL are 1200 for LD operation first. Then 7556 for ADD operation, d220 for INC operation, e440 for DEC operation, f428 for BNE operation. If the process satisfies the Z=0 condition at this point, it returns to the beginning of the LABEL and continues the process from there. If the Z=0 condition is not accepted, the register we keep as the counter is equal to zero and the process exits the loop of LABEL and is d220 for INC, It receives 2104 commands for ST and finishes the process. As we can see in Figure 23, the sum of the values a0 a1 a2 a3 a4 is written at the a6 th address.

00	20 04 00 00	00 00 00 00	00 00 00 00	00 00 00 00
10	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
20	05 14 00 15	a0 16 60 32	00 12 56 75	20 d2 40 e4
30	28 f4 20 d2	04 21 00 00	00 00 00 00	00 00 00 00
40	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
50	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
60	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
70	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
80	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
90	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
a0	01 02 03 04	05 00 0f 00	00 00 00 00	00 00 00 00
b0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
c0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
d0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
e0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
f0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

Figure 27: Test in Memory (RAM)

4 DISCUSSION

During the project, Firstly we divided IR to ADDRESS, REGSEL, ADDRESSING MODE, OPCODE and secondly again we divided IR to OPCODE, DESTREG, SRCREG1, SRCREG2 to get proper Instruction. After we designed Fetch and Decode circuits. We decoded REGSEL to Rx, DESTREG to Register File RegSel (changeable RegSel of RF according to Instructions). We multiplexed DESTREG to get the correct RegSel(ARF). Similarly we multiplexed OPCODE to get the proper FunSel(ALU) and multiplexed OPCODE again to get INC DEC FunSel which we used in INC DEC opera-

tions. Finally we designed operation circuits. In the and of the project, we have learned to create a control unit circuit, how memory works, how logisim tunnels uses, designing sequential counter and implement operations with correct order.

5 CONCLUSION

As a consequence of this project, we learnt a lot about the technical part of a basic computer by applying a lot of logistical instructions, and we also learned much about essential CPUs.