Faculty of Engineering & Technology Electrical & Computer Engineering Department

ENCS2380: Computer Organization and Microprocessor

Single Cycle Processor Design Project Report

Prepared by:

Taleed Mahmoud Hamadneh 1220006

Qasim Nidal Batrawi 1220204

Instructor: Dr. Ismail Khater

Section: 1

Date: 9 June 2024

Table of Contents

Register File

The register file contains seven 32 bit registers, R0 is connected to zero that is the reading from it returns zero value and the writing on it has no effect. However, R1 to R7 are required for the machine.

This file has one write port (Rx) which refers to the destination register and two read ports (Ry & Rz) that are considered as the sources registers.

To implement the above design, We have used 3 3 to 8 Decoder, 7 AND gates, 7 registers, 16 tri buffer states, one clock, and one RegWrite input to activate the writing process.

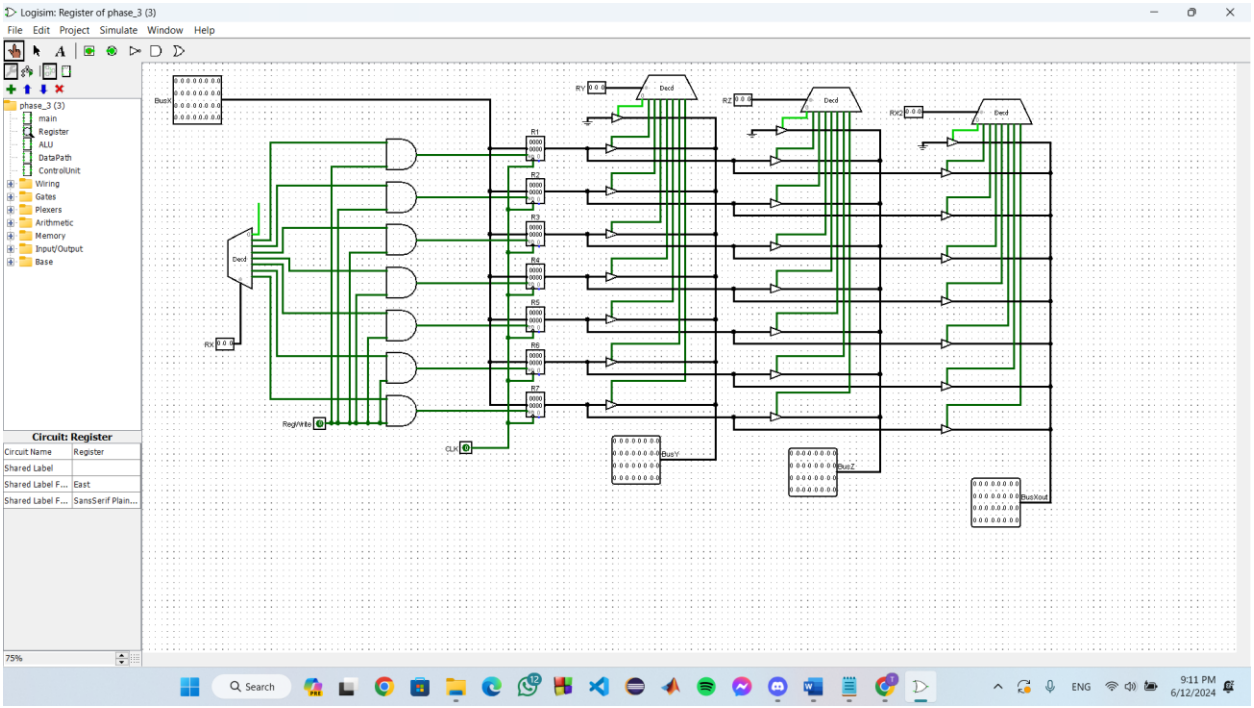

Figure 1 Register File Circuit

As shown in figure 1, the main decoder decodes the wanted register to write on specified on the 3 bit input (001 -> R1, 111-> R7 ….). Each register take 2 inputs: one for the clock and the other is the data to be written (from bus x). Furthermore, the output has been chosen based on the 2 decoders which choose the needed register based on the 3 bits each (RY & RZ).

Note that we have added a new output for bus x, which was needed for branch instructions that have x in it's format.

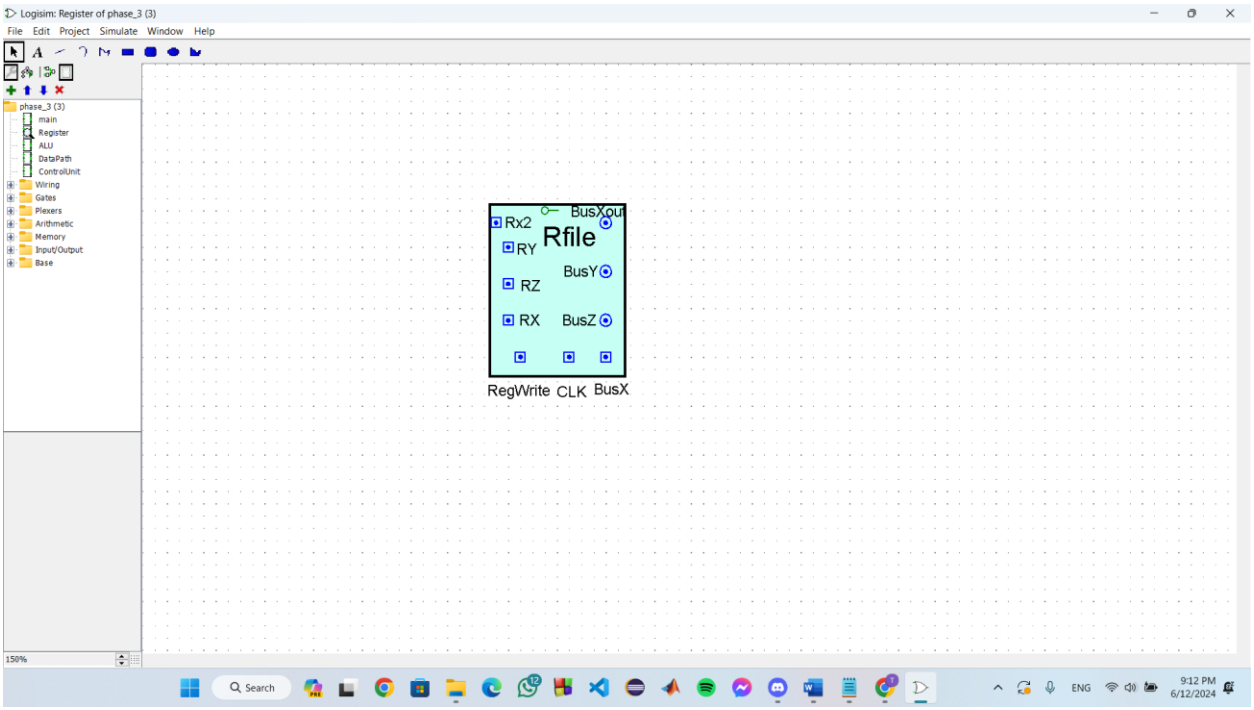

Figure 2 Register File Block

ALU

This processor has a 32 bit ALU to handle the following operations:

XOR, AND, OR, CAND, ADD, SEQ, NADD, SLT, SRA, SRL, SLL, and ROR.

As shown in figure 2 & figure 3, We have used 16x1 MUX, with 4 unneeded inputs.

** The Table below describes the functionality of the ALU

Input no.	Operation	How
0	XOR	Basic XOR gate
1	AND	Basic AND gate
2	OR	Basic OR gate
3	CAND	Not & AND gates
4	ADD	32 bit adder Block
5	SEQ	Basic XOR gate

6	NADD	32 bit subtractor
7	SLT	Not & AND gates
8	SRA	Right Arithmetic Shifter & splitter(5bits)
9	SRL	Right Logical shifter & splitter(5bits)
10	SLL	Left Logical shifter & splitter (5 bits)
11	ROR	Right rotator & splitter (5 bits)

2 operands are required (A & B), and 4 bit selector to choose the operation.

Note that we have made 2 forms of the ALU, one of them contains a bitwise comparator using the basic gates (XOR and AND gates), and the other contains a 32 bit comparator that returns only 1 or 0 value.

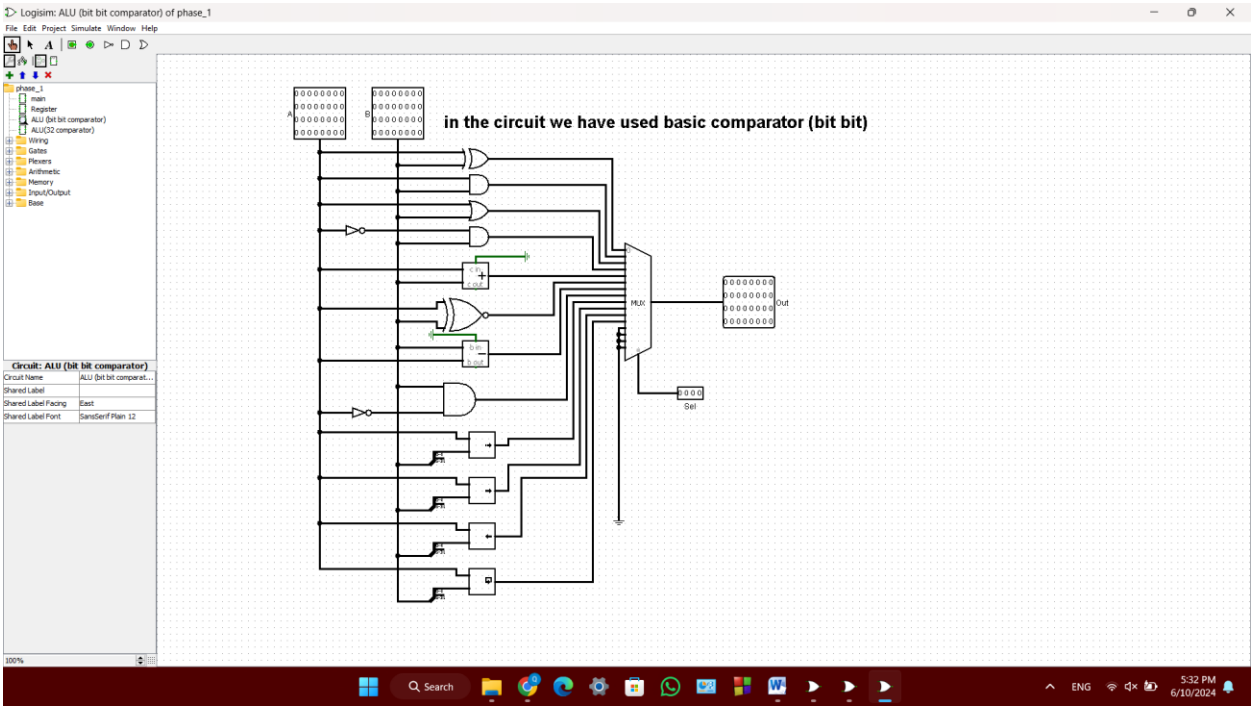

Figure 3 ALU circuit 1

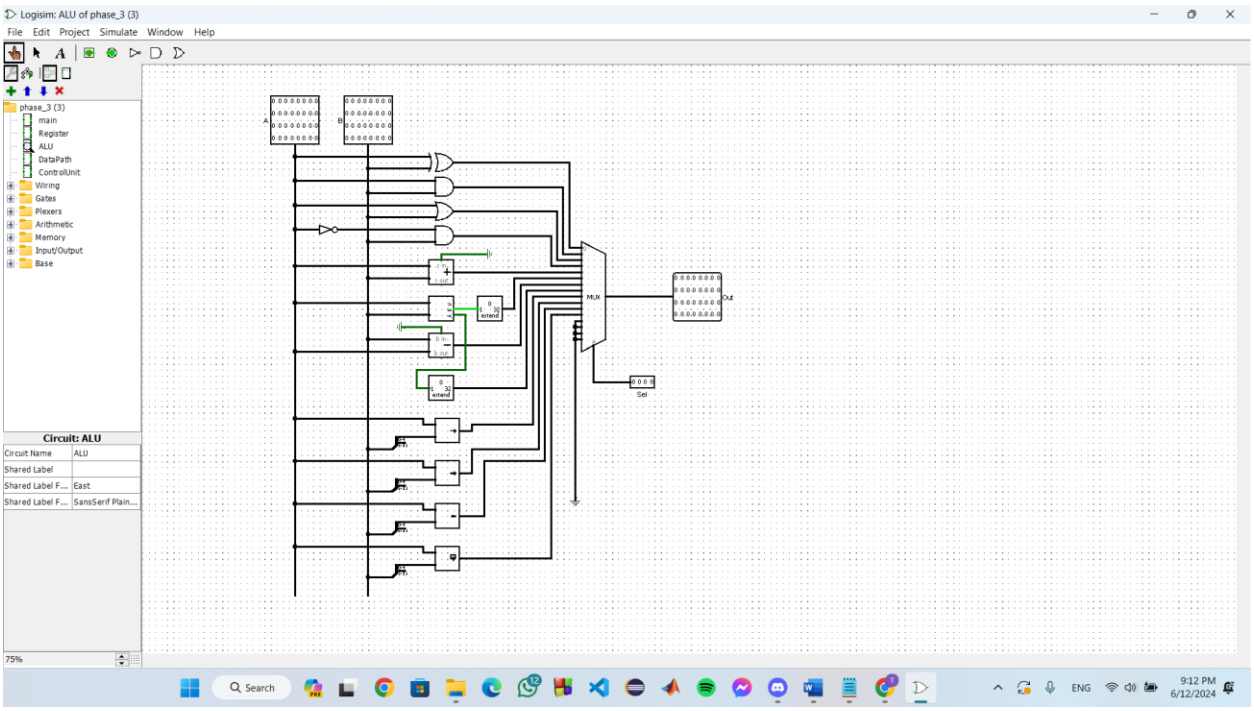

Figure 4 ALU circuit 2

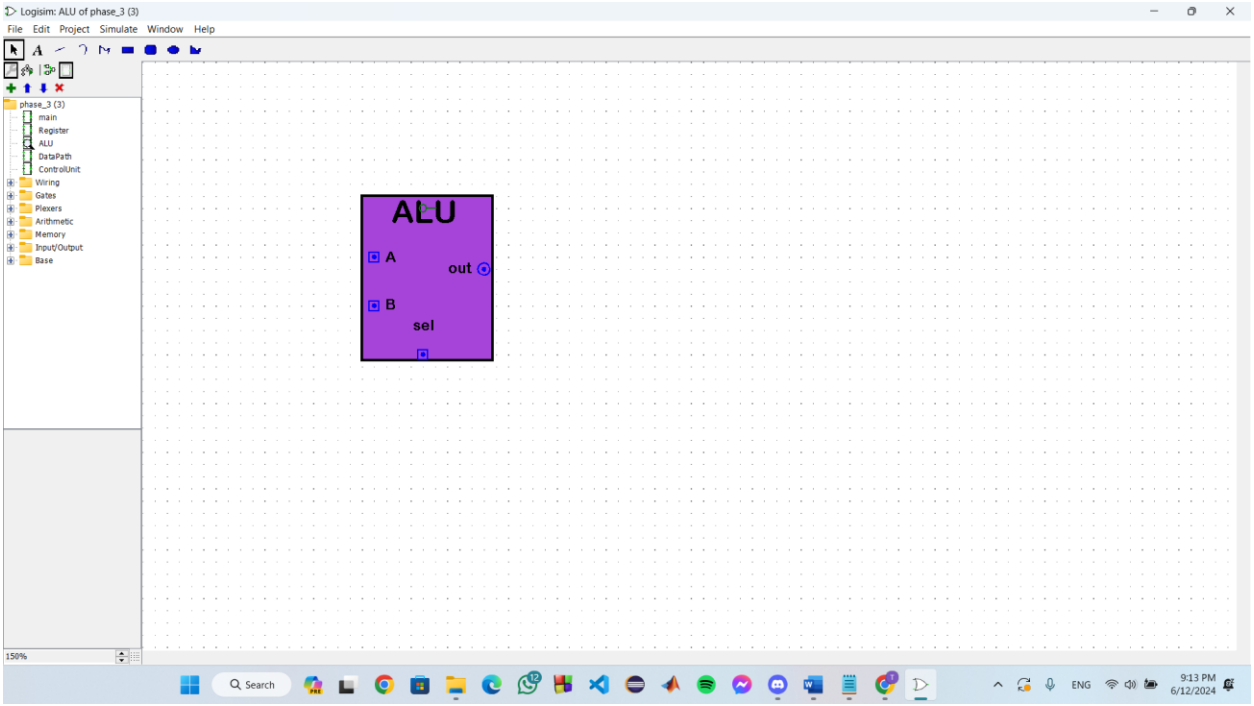

Figure 5 ALU Block

Data Path

The Data Path has 5 major parts (PC register, Instruction Memory, Register File, ALU, Data Memory).

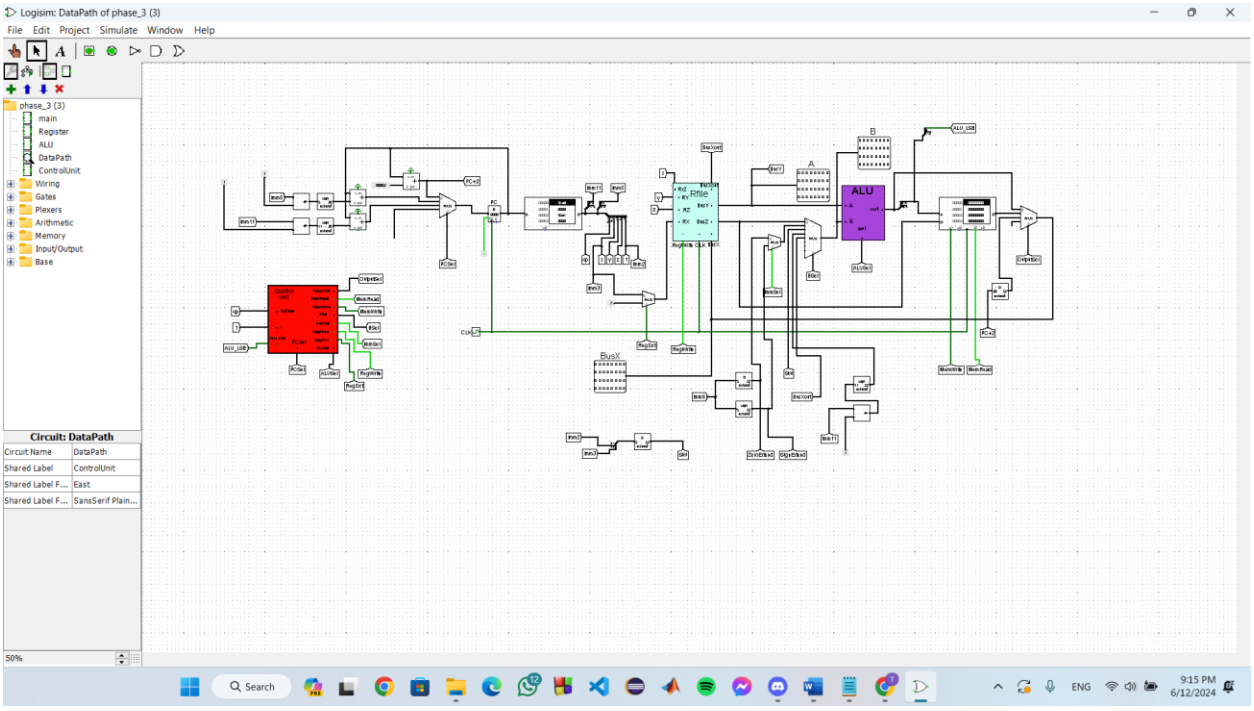

Figure 6 DataPath Circuit

1- PC register:

The PC register has 3 inputs: one for the clock, one for enable (always 1) and the other is for the data of the PC. To implement that we have used a 8x1 mux with 3 unneeded inputs, Input 0 has the content of PC+2 which is the normal increment to the next instruction. Input 1 has PC + sign extend to the Immediate 5 after shifting it one bit to the left. Input 2 indicates to PC + sign extend to the Immediate 11 after shifting it one bit to the left.

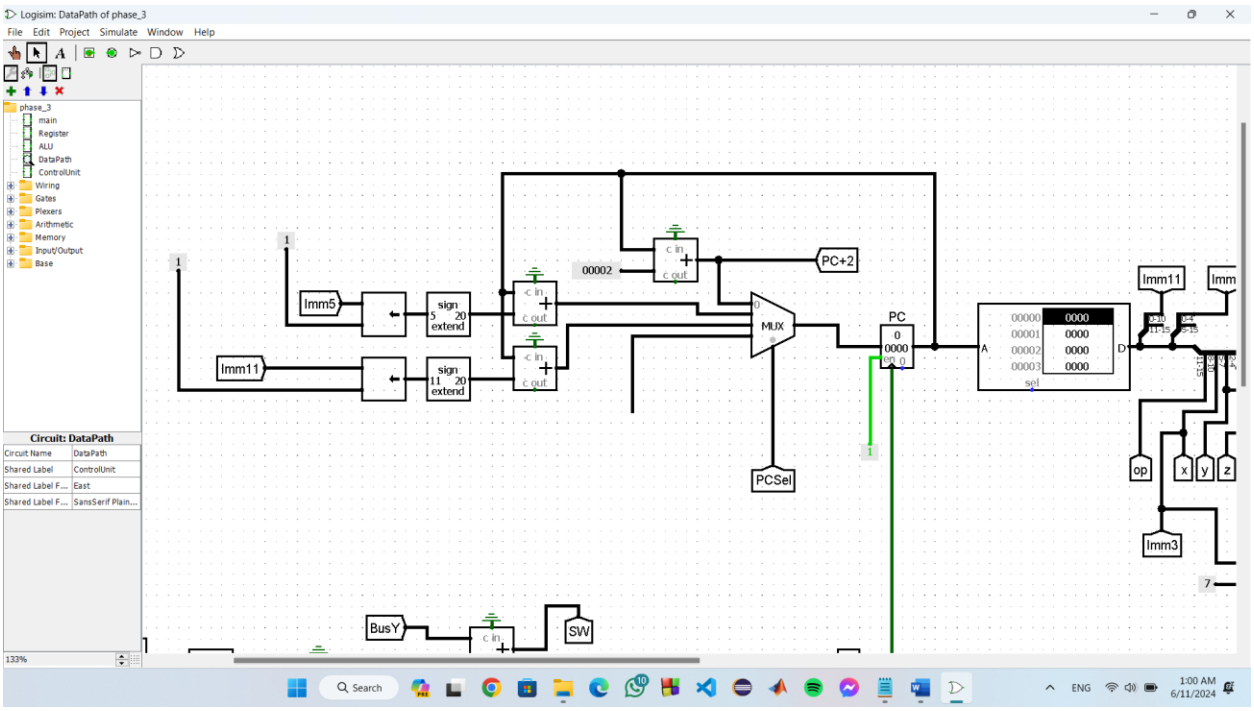

Figure 7 PC Register

2- Instruction Memory

ROM: Which receives 20 bit from the PC for the address width (Because the PC is 20 bit), and 16 Bit for the data width (The instructions are 16 bits). It's outputs cover the four instruction types, R-type, I-type, S-type, and J-type. We divided the 16 bit by splitters to obtain : Imm11, imm5, x, y, z, f, imm2, opcode, and imm3.

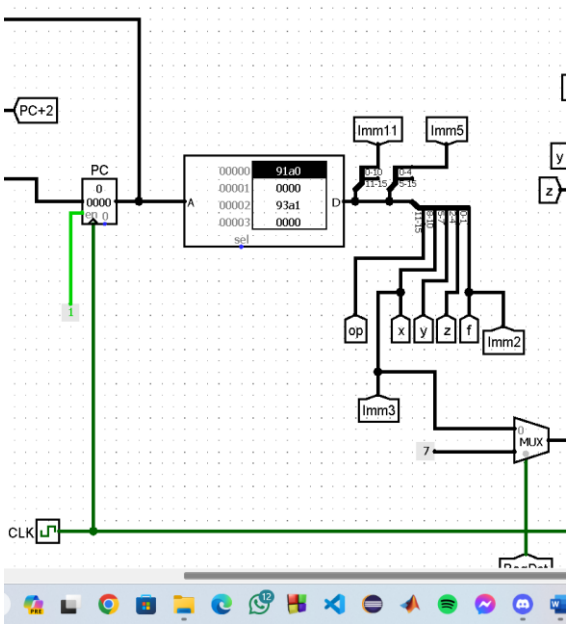

Figure 8 Instruction memory

3- Data Memory

RAM: Which receives 20 bit from the ALU by a splitter for the address width and 32 Bit for the data width. It needs 2 control signals to determine writing on or reading from the memory. Moreover, it needs 2 inputs: One from the alu output, and the other from Bus z. it's output enter to a mux to determine the destination of any operation: 00 -> the ALU output, 01 ->Memory read, 10 -> PC+2

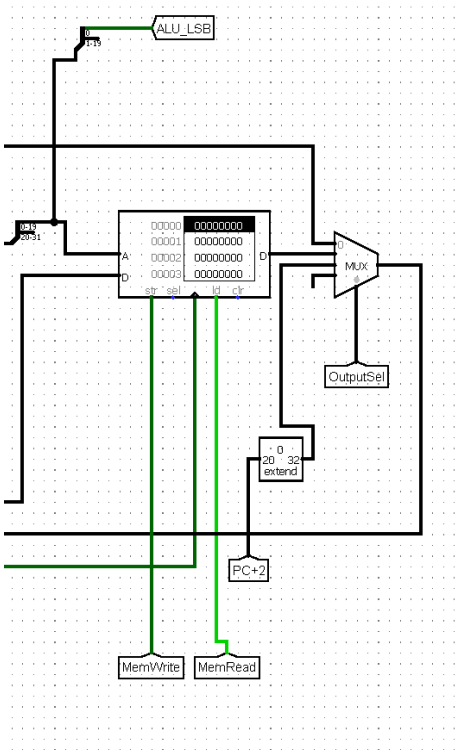

Figure 9 Data Memory

4- Rfile

It needs two main control signals: RegWrite which enables the process of writing on a specific register, and CLK. Also, it takes 4 selection inputs: Ry and RZ and Rx for the sources registers, and RX for the destination one (there was a mux to determine the wanted destination register which can be either R7 or Rx. It outs two outputs: Bus Y and Bus z which will enter to the ALU later.

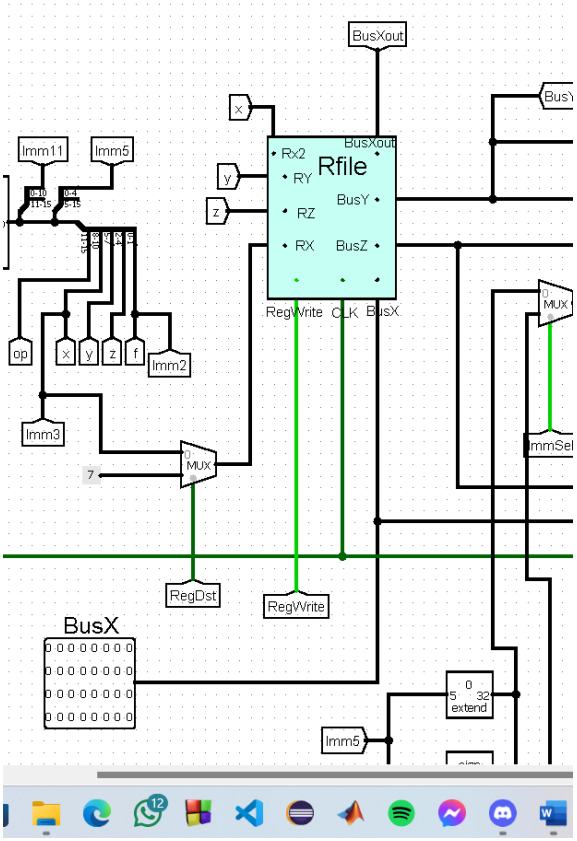

Figure 10 R File

5- ALU:

As known, it takes two opperands: Bus Y and Mux with one of these operands {

1-Bus Z

2-IMM5 -> sign extend

3-IMM5-> zero extend

4-{IMM3,IMM2}

5-BusXout (used for branch instructions)

}

All above operands are selected by a mux based on BSEL signal.

Also, there is one control signal to determine the right operation to be executed in the ALU (ALUSEL).

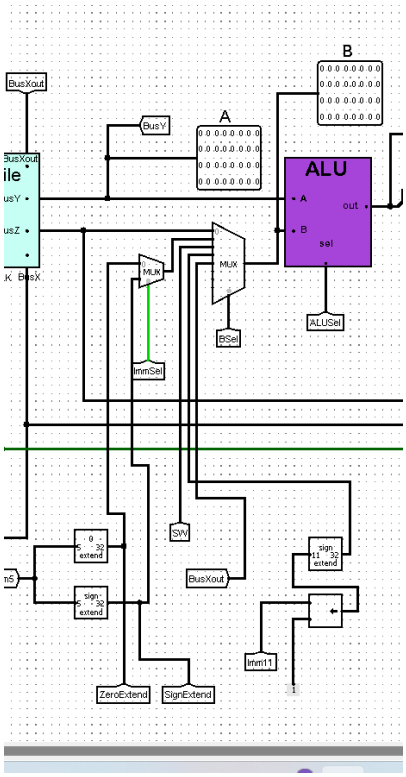

Figure 11 ALU

Control Unit

The control unit receives 3 inputs which are the opcode, f and the LSB of the ALU result, Here's how each input operates:

1- OpCode: It will be used in all type of instructions; it will also determine the operation to be done.

2- f: It will be used in the R-type instruction, and it will also help the OpCode to choose the operation to be done.

3- ALU_LSB: it will be used in the I-type instruction, specifically in the branches instructions, it will determine the increment type for the program counter.

In the data path, there are several control signals which will determine the behavior of the components of the data path. These control signals are:

1- **PCSel**: Controls the Program Counter (PC) to choose the type of increment.
2- **RegDst**: Chooses the register that will be considered as a destination.
3- **RegWrite**: Determines whether to write data on the destination register or not (enable when writing on register).
4- **ImmSel**: Chooses the type of extension for the immediate (Imm5).
5- **BSel**: Selects the second input of the ALU.
6- **ALUSel**: Determines the type of operation to be performed by the ALU.
7- **MemWrite**: Controls whether to write data on the memory or not.
8- **MemRead**: Controls whether to read data from the memory or not.
9- **OutputSel**: Chooses the data to be stored in the destination register.

These control signals are generated depending on the value of the **Opcode** , f, and **ALU_LSB.**

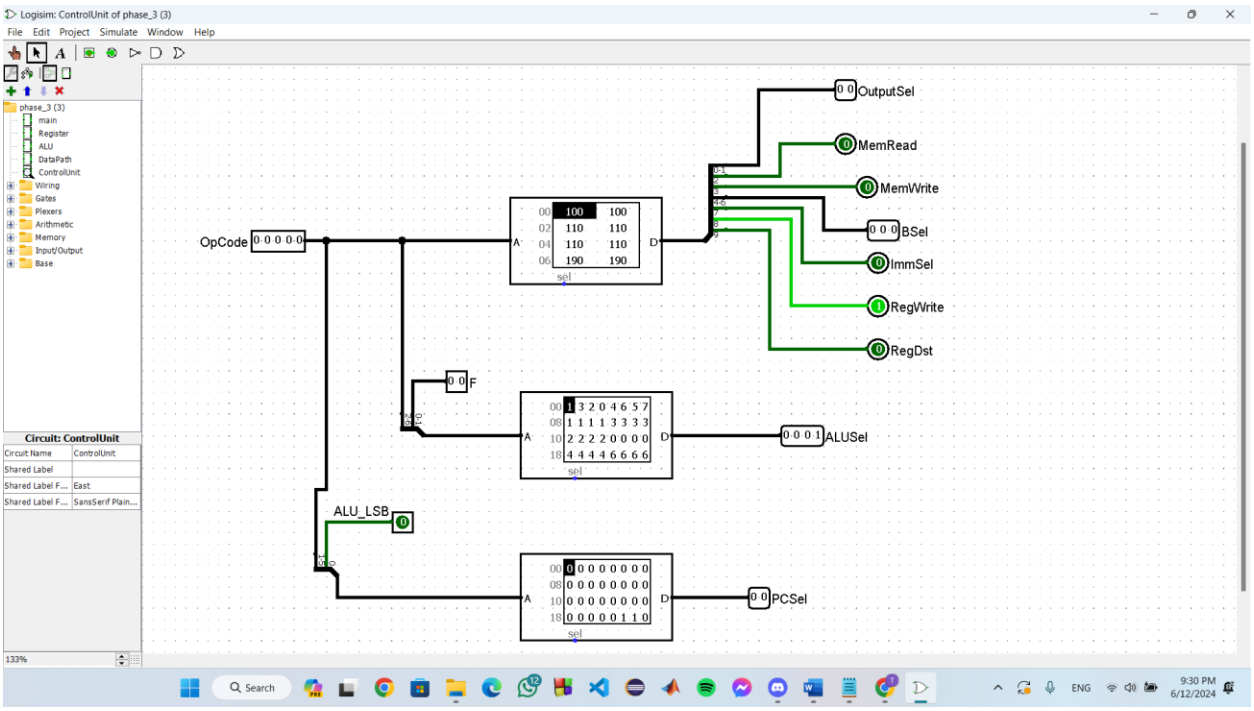

Figure 12 Control unit circuit

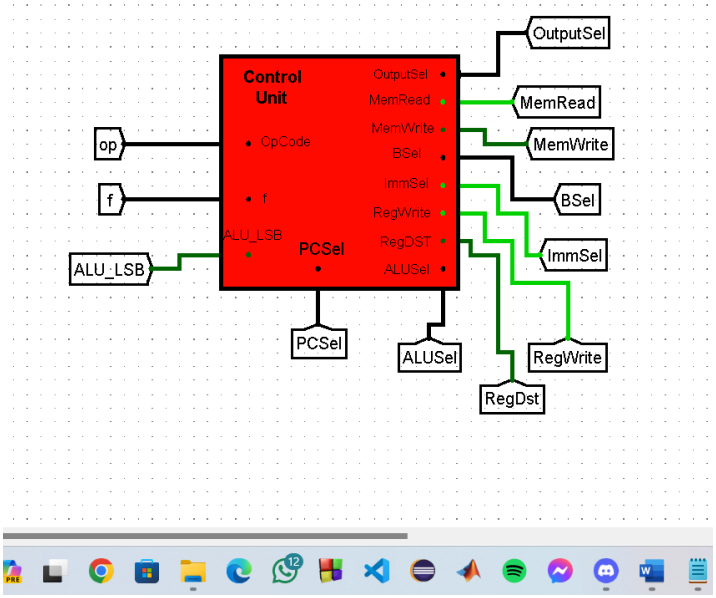

Figure 13 Control unit Block

The following table defines the values of RegDst, RegWrite, ImmSel, BSel, ALUSel, MemWrite, MemRead and OutputSel.

Opcode	Control Signals	Operation
0x00	0x100	AND
0x00	0x100	CAND
0x00	0x100	OR
0x00	0x100	XOR
0x01	0x100	ADD
0x01	0x100	NADD
0x01	0x100	SEQ
0x01	0x100	SLT
0x02	0x110	ANDI
0x03	0x110	CANDI
0x04	0x110	ORI
0x05	0x110	XORI
0x06	0x190	ADDI
0x07	0x190	NADDI
0x08	0x190	SEQI
0x09	0x190	SLTI
0x0A	0x110	SLL
0x0B	0x110	SRL
0x0C	0x110	SRA
0x0D	0x110	ROR
0x0E	0x040	BEQ
0x0F	0x040	BNE
0x10	0x040	BLT
0x11	0x040	BGE
0x12	0x195	LW
0x13	0x02B	SW
0x14	0x073	J
0x15	0x372	JAL

Table 1 Control Signals

The following table defines the ALUSel values, which are used to select the type of the operation that the ALU will perform.

Opcode	f	ALUSel	ALU Operation	Location
0x00	0x0	0x1	AND	0x00
0x00	0x1	0x3	CAND	0x01
0x00	0x2	0x2	OR	0x02
0x00	0x3	0x0	XOR	0x03
0x01	0x0	0x4	ADD	0x04
0x01	0x1	0x6	NADD	0x05
0x01	0x2	0x5	SEQ	0x06
0x01	0x3	0x7	SLT	0x07
0x02	0x0	0x1	AND	0x08
0x02	0x1	0x1	AND	0x09
0x02	0x2	0x1	AND	0x0A
0x02	0x3	0x1	AND	0x0B
0x03	0x0	0x3	CAND	0x0C
0x03	0x1	0x3	CAND	0x0D
0x03	0x2	0x3	CAND	0x0E
0x03	0x3	0x3	CAND	0x0F
0x04	0x0	0x2	OR	0x10
0x04	0x1	0x2	OR	0x11
0x04	0x2	0x2	OR	0x12
0x04	0x3	0x2	OR	0x13
0x05	0x0	0x0	XOR	0x14
0x05	0x1	0x0	XOR	0x15
0x05	0x2	0x0	XOR	0x16
0x05	0x3	0x0	XOR	0x17
0x06	0x0	0x4	ADD	0x18
0x06	0x1	0x4	ADD	0x19
0x06	0x2	0x4	ADD	0x1A
0x06	0x3	0x4	ADD	0x1B
0x07	0x0	0x6	NADD	0x1C
0x07	0x1	0x6	NADD	0x1D
0x07	0x2	0x6	NADD	0x1E
0x07	0x3	0x6	NADD	0x1F

0x08	0x0	0x5	SEQ	0x20
0x08	0x1	0x5	SEQ	0x21
0x08	0x2	0x5	SEQ	0x22
0x08	0x3	0x5	SEQ	0x23
0x09	0x0	0x7	SLT	0x24
0x09	0x1	0x7	SLT	0x25
0x09	0x2	0x7	SLT	0x26
0x09	0x3	0x7	SLT	0x27
0x0A	0x0	0xA	SLL	0x28
0x0A	0x1	0xA	SLL	0x29
0x0A	0x2	0xA	SLL	0x2A
0x0A	0x3	0xA	SLL	0x2B
0x0B	0x0	0x9	SRL	0x2C
0x0B	0x1	0x9	SRL	0x2D
0x0B	0x2	0x9	SRL	0x2E
0x0B	0x3	0x9	SRL	0x2F
0x0C	0x0	0x8	SRA	0x30
0x0C	0x1	0x8	SRA	0x31
0x0C	0x2	0x8	SRA	0x32
0x0C	0x3	0x8	SRA	0x33
0x0D	0x0	0xB	ROR	0x34
0x0D	0x1	0xB	ROR	0x35
0x0D	0x2	0xB	ROR	0x36
0x0D	0x3	0xB	ROR	0x37
0x0E	0x0	0x5	SEQ	0x38
0x0E	0x1	0x5	SEQ	0x39
0x0E	0x2	0x5	SEQ	0x3A
0x0E	0x3	0x5	SEQ	0x3B
0x0F	0x0	0x5	SEQ	0x3C
0x0F	0x1	0x5	SEQ	0x3D
0x0F	0x2	0x5	SEQ	0x3E
0x0F	0x3	0x5	SEQ	0x3F
0x10	0x0	0x7	SLT	0x40
0x10	0x1	0x7	SLT	0x41
0x10	0x2	0x7	SLT	0x42
0x10	0x3	0x7	SLT	0x43
0x11	0x0	0x7	SLT	0x44
0x11	0x1	0x7	SLT	0x45
0x11	0x2	0x7	SLT	0x46

0x11	0x3	0x7	SLT	0x47
0x12	0x0	0x4	ADD	0x48
0x12	0x1	0x4	ADD	0x49
0x12	0x2	0x4	ADD	0x4A
0x12	0x3	0x4	ADD	0x4B
0x13	0x0	0x4	ADD	0x4C
0x13	0x1	0x4	ADD	0x4D
0x13	0x2	0x4	ADD	0x4E
0x13	0x3	0x4	ADD	0x4F
0x14	0x0	0x0	XOR	0x50
0x14	0x1	0x0	XOR	0x51
0x14	0x2	0x0	XOR	0x52
0x14	0x3	0x0	XOR	0x53
0x15	0x0	0x0	XOR	0x54
0x15	0x1	0x0	XOR	0x55
0x15	0x2	0x0	XOR	0x56
0x15	0x3	0x0	XOR	0x57

Table 2 ALU selectors

NOTES:

1- Last 8 ALU operations in the table above are actually “don’t care”, since we don’t have to perform any operation when the **Opcode** is 20 or 21.

2- For I-type, S-type and J-type instructions, the value of **f** will be “don’t care”, so all possible cases for **f** must be considered for these instructions.

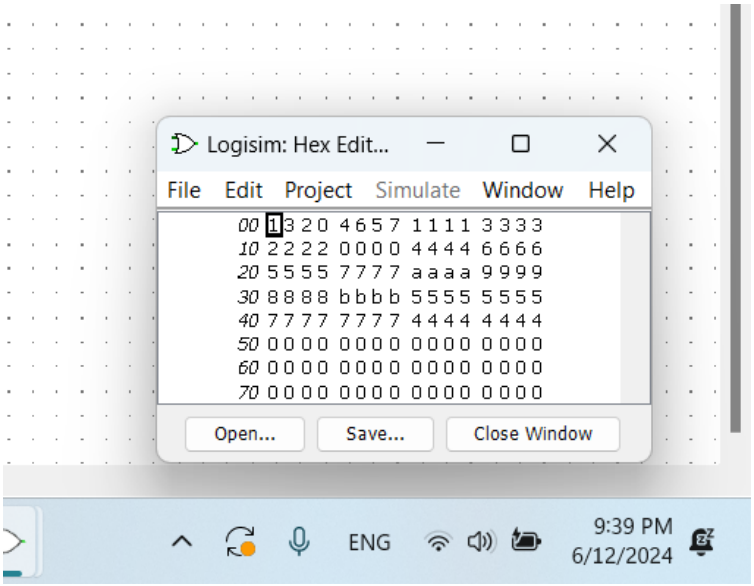

Figure 14 ALUSel ROM

The following table defines the PCSel values, which are used to select the type of increment for the Program Counter.

Opcode	ALU LSB	Location on memory	PC increment type	PC selector
0x00	0x0	0x00	PC+=2	00
0x00	0x1	0x01	PC+=2	00
0x01	0x0	0x02	PC+=2	00
0x01	0x1	0x03	PC+=2	00
0x02	0x0	0x04	PC+=2	00
0x02	0x1	0x05	PC+=2	00
0x03	0x0	0x06	PC+=2	00
0x03	0x1	0x07	PC+=2	00
0x04	0x0	0x08	PC+=2	00
0x04	0x1	0x09	PC+=2	00
0x05	0x0	0x0A	PC+=2	00
0x05	0x1	0x0B	PC+=2	00
0x06	0x0	0x0C	PC+=2	00
0x06	0x1	0x0D	PC+=2	00
0x07	0x0	0x0E	PC+=2	00

0x07	0x1	0x0F	PC+=2	00
0x08	0x0	0x10	PC+=2	00
0x08	0x1	0x11	PC+=2	00
0x09	0x0	0x12	PC+=2	00
0x09	0x1	0x13	PC+=2	00
0x0A	0x0	0x14	PC+=2	00
0x0A	0x1	0x15	PC+=2	00
0x0B	0x0	0x16	PC+=2	00
0x0B	0x1	0x17	PC+=2	00
0x0C	0x0	0x18	PC+=2	00
0x0C	0x1	0x19	PC+=2	00
0x0D	0x0	0x1A	PC+=2	00
0x0D	0x1	0x1B	PC+=2	00
0x0E	0x0	0x1C	PC+=2	00
0x0E	0x1	0x1D	PC+=Imm5	01
0x0F	0x0	0x1E	PC+=Imm5	01
0xFF	0x1	0x1F	PC+=2	00
0x10	0x0	0x20	PC+=2	00
0x10	0x1	0x21	PC+=Imm5	01
0x11	0x0	0x22	PC+=Imm5	01
0x11	0x1	0x23	PC+=2	00
0x12	0x0	0x24	PC+=2	00
0x12	0x1	0x25	PC+=2	00
0x13	0x0	0x26	PC+=2	00
0x13	0x1	0x27	PC+=2	00
0x14	0x0	0x28	PC+=Imm11	10
0x14	0x1	0x29	PC+=Imm11	10
0x15	0x0	0x2A	PC+=Imm11	10
0x15	0x1	0x2B	PC+=Imm11	10

Table 3 PC Selector

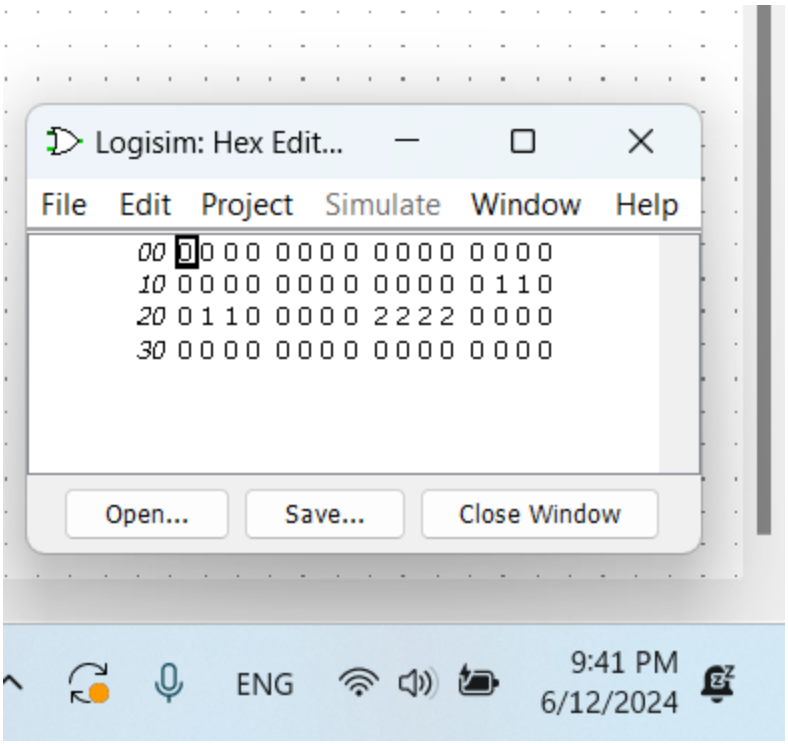

Figure 14 PC Sel ROM

Simulation and Testing part

1- LOAD LW

First of all, we have used the load instruction 0x9141 which refers to:

LW R1, [R2+2_00001]

Explanation: The value stored in the memory location was loaded on R1. R2 is used to determine the memory location (R2= R1 + sign extend imm5). Also, the memory has the data 0x00000011 on the first ten addressees.

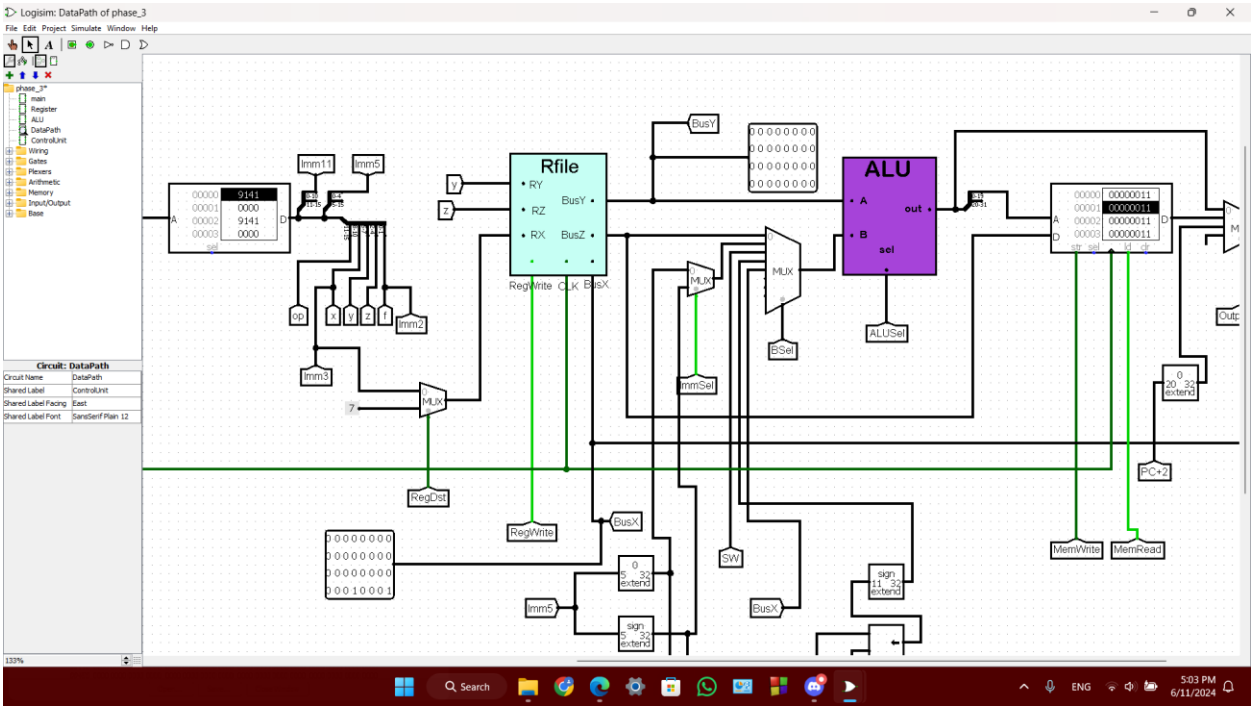

Figure 15 Load instruction

Then, we have loaded another value from the memory and stored it on R2 as the above figure to implement the rest operations.

Now, R1 and R2 has the value 0x00000011.

2-AND

To operate AND operation, we have used 0x0628 instruction which refers to:

AND R6, R1, R2

Explanation: Bitwise and between two same values (R1, R2), so the answer was the same = 0x00000011 as shown below.

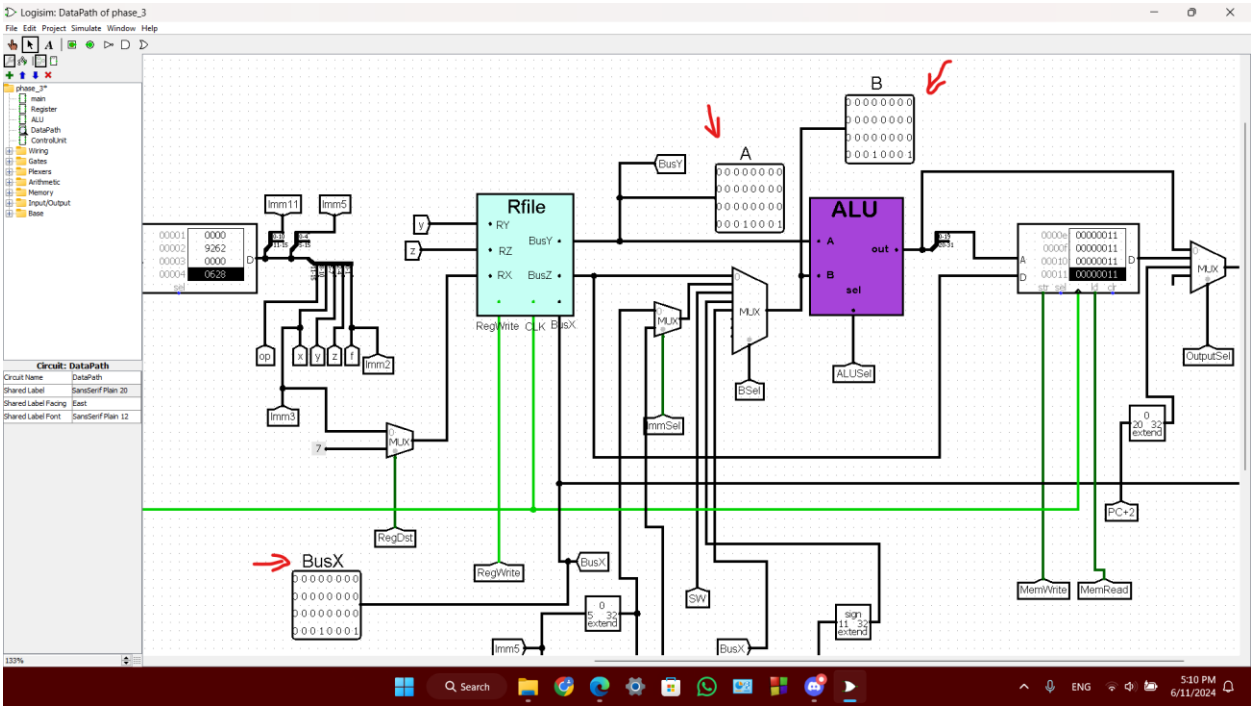

Figure 16 AND instruction

3-CAND

To operate the and operation, we have used 0x0729 instruction which refers to:

CAND R7, R1, R2

Explanation: While R1 and R2 have the same values, CAND must be zero (R1 was negated inside the ALU then AND with R2 resulting zero).

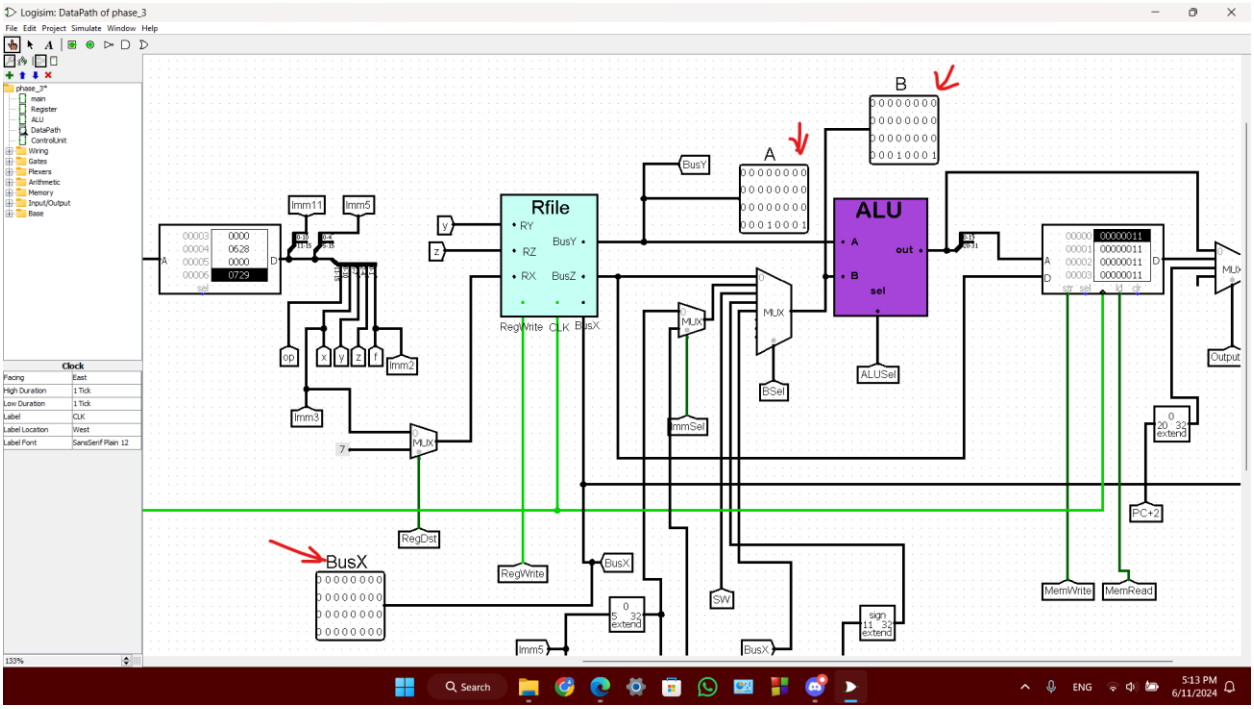

Figure 17 CAND instruction

4-OR

0x042A was used to test the bitwise or:

OR R4, R1, R2

Explanation: Bitwise or between two same values (R1, R2), so the answer was the same = 0x00000011 as shown below.

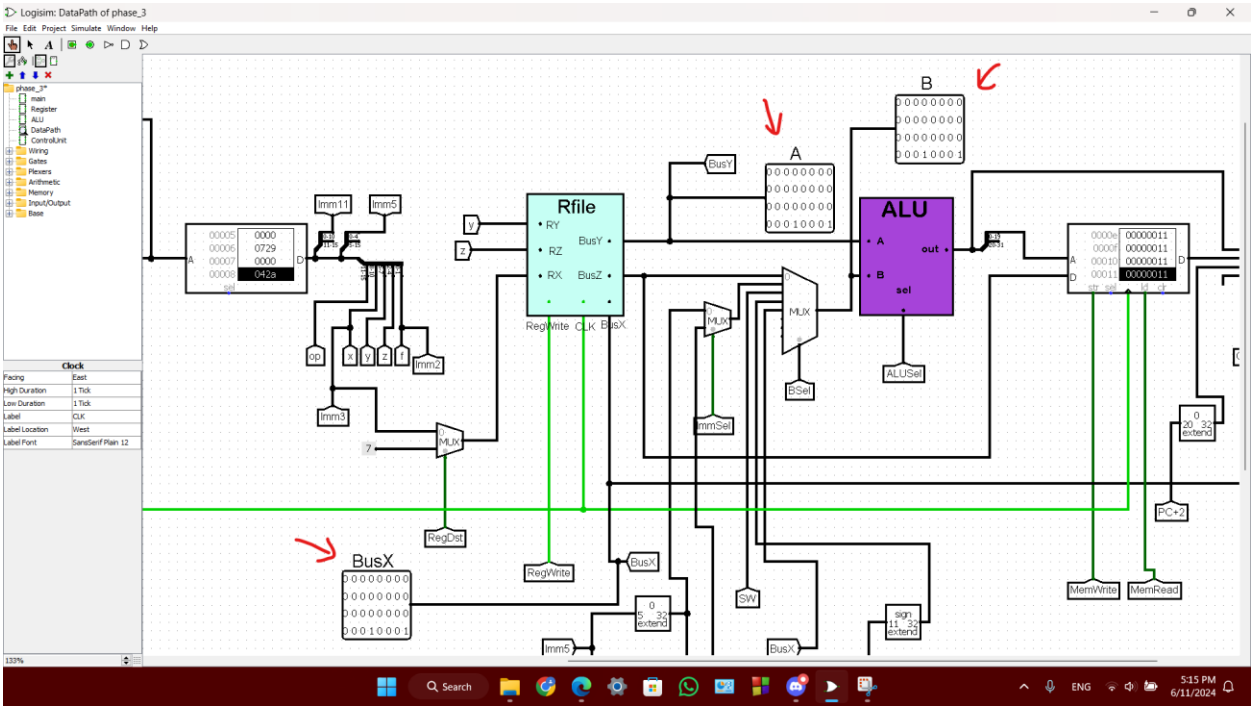

Figure 18 OR instruction

5-XOR

To avoid repetition in results, we have loaded new values on R2 by 0x9260 instruction:

LW R2,[R3+2_00000]

So, now we have R1=0x00000011 and R2=0x00000111

The instruction = 0x062B

XOR R6, R1, R2

Result was shown in BusX below.

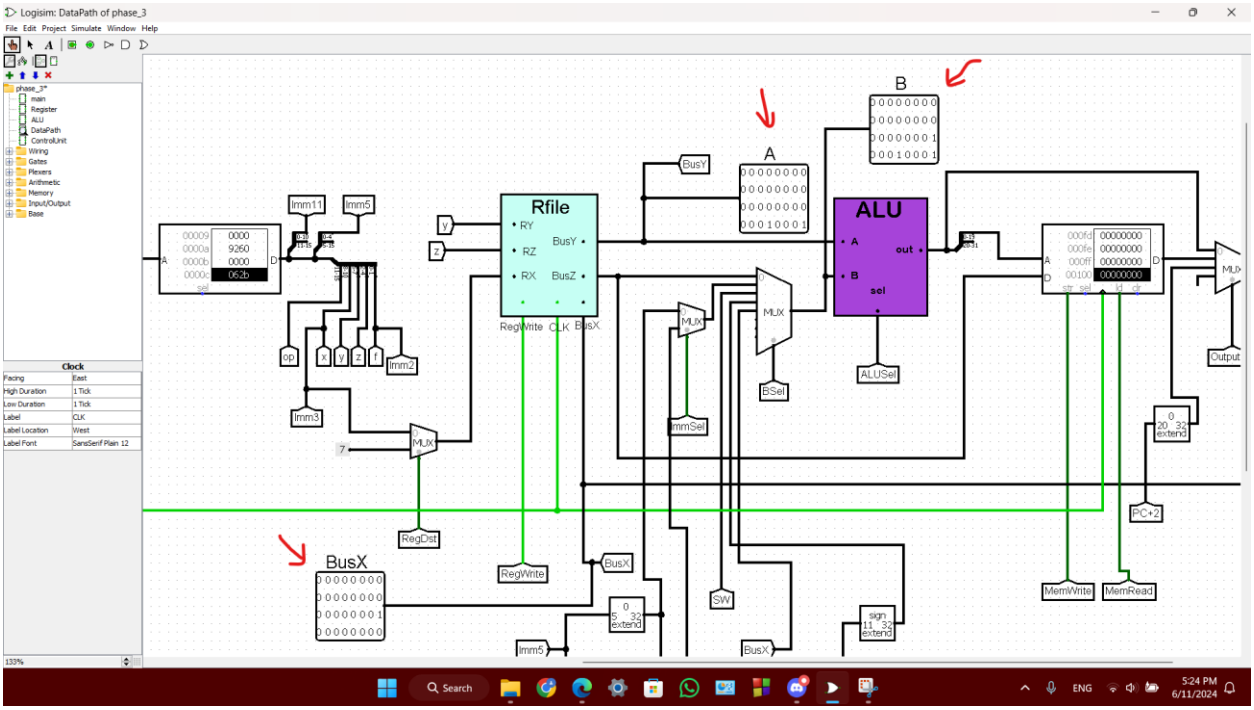

Figure 19 XOR instruction

6-ADD

The instruction is 0x0E28.

ADD R6, R1, R1

R1=0x00000011

R2=0x00000111

R6=0x00000122 as shown in Bus x below:

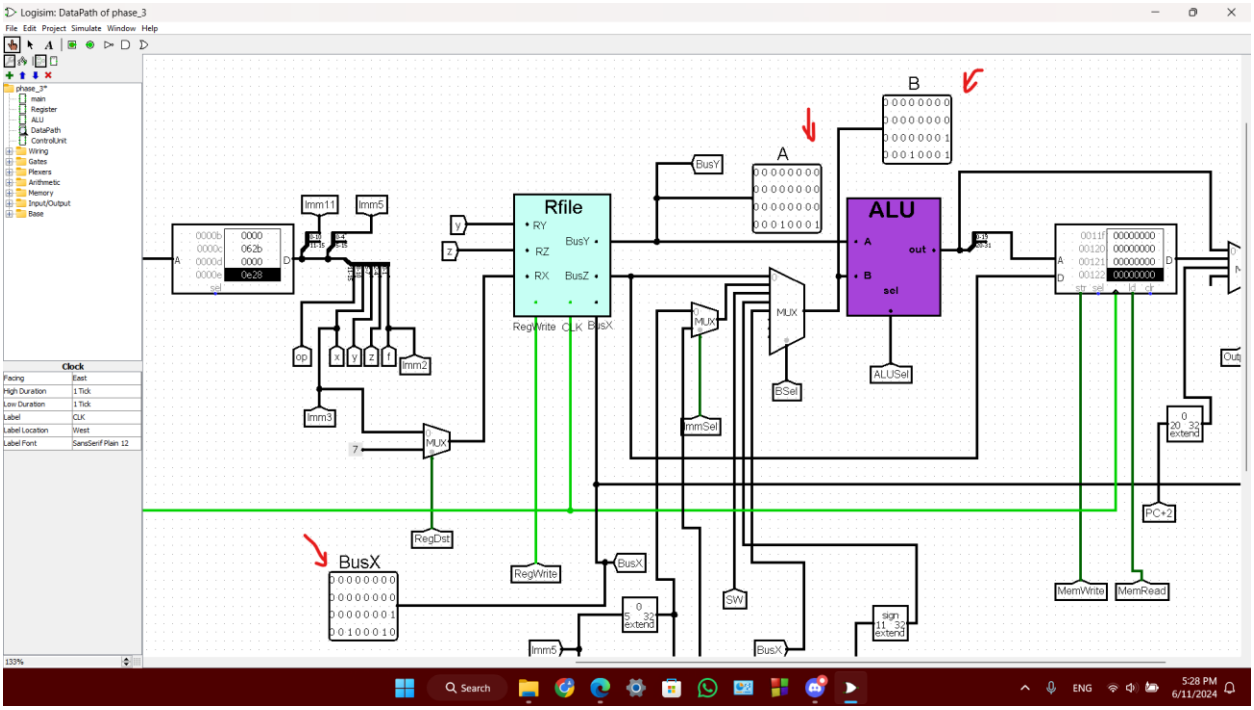

Figure 20 ADD instruction

7-NADD

The instruction is 0x0E29.

NADD R6, R1, R2

R1=0x00000011

R2=0x00000111

The result is shown below in BusX.

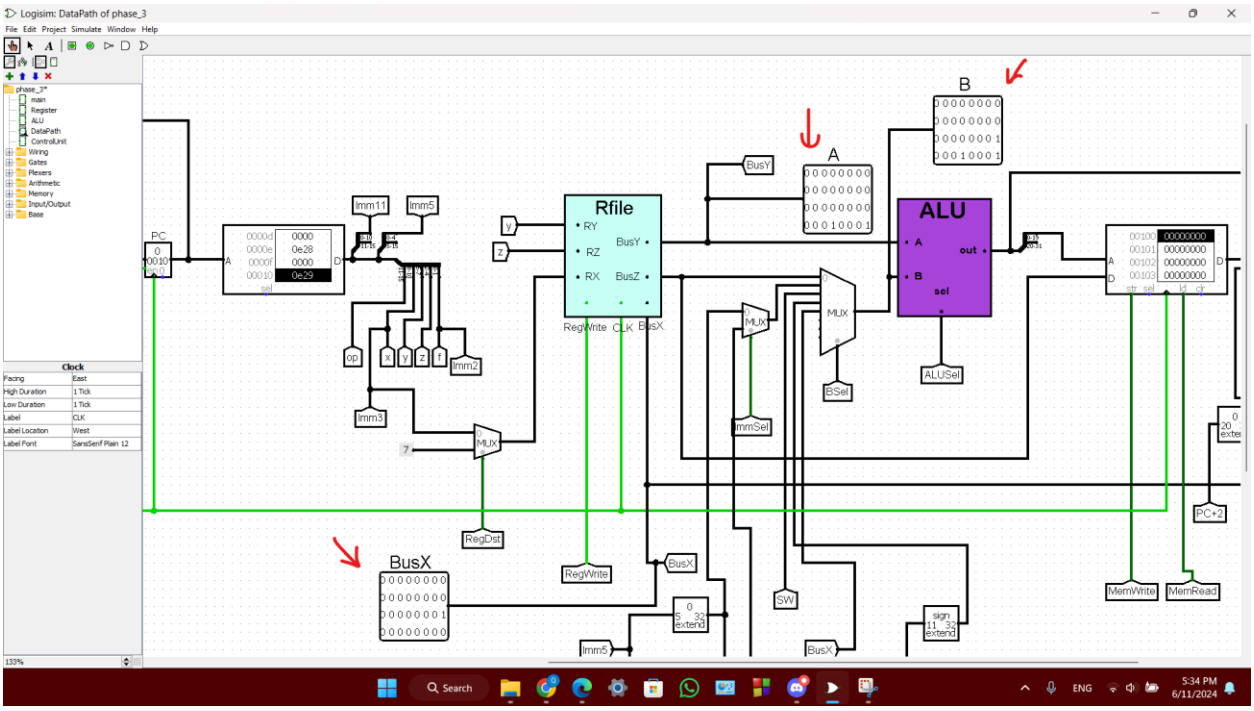

Figure 21 NADD instruction

8-SEQ

The instruction is 0x0F2A.

SEQ R7,R1,R2

Since R1 and R2 are not equal, the value will be zero, which was shown below.

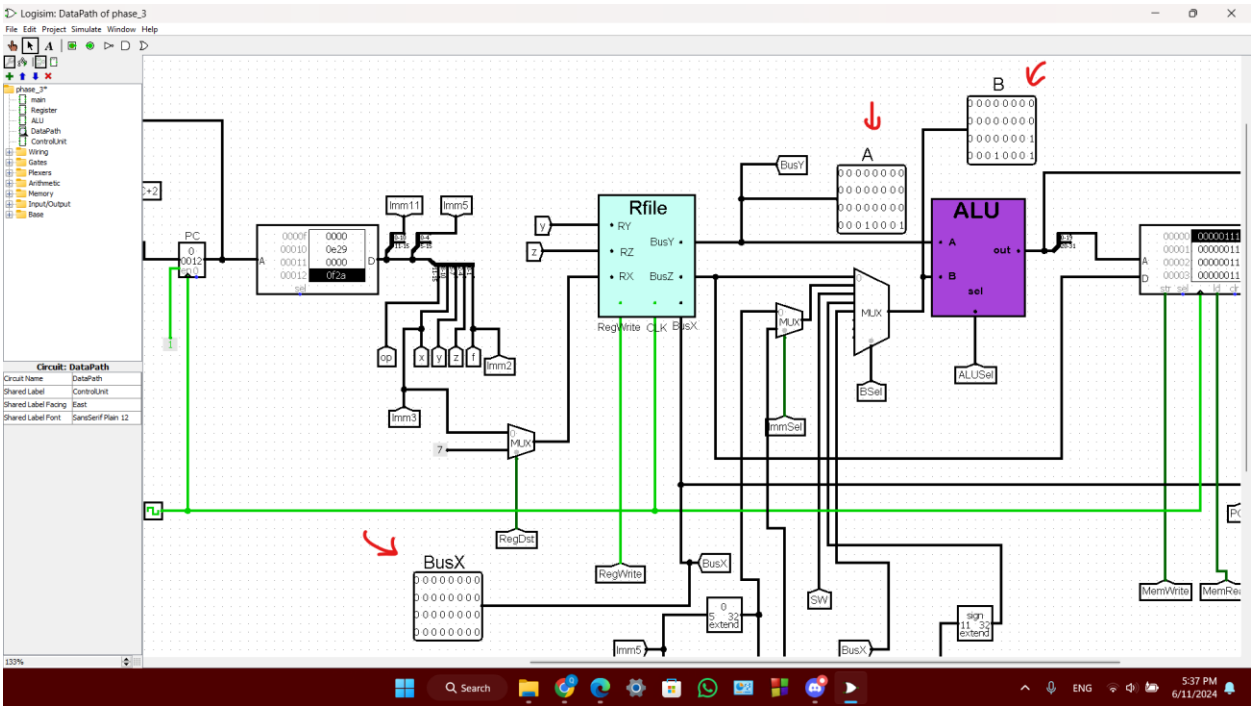

Figure 22 SEQ instruction

9-SLT

The instruction is 0x0F2B.

SLT R7,R1,R2

R1=0x00000011

R2=0x00000111

Since R2 > R1 the result must be 1.

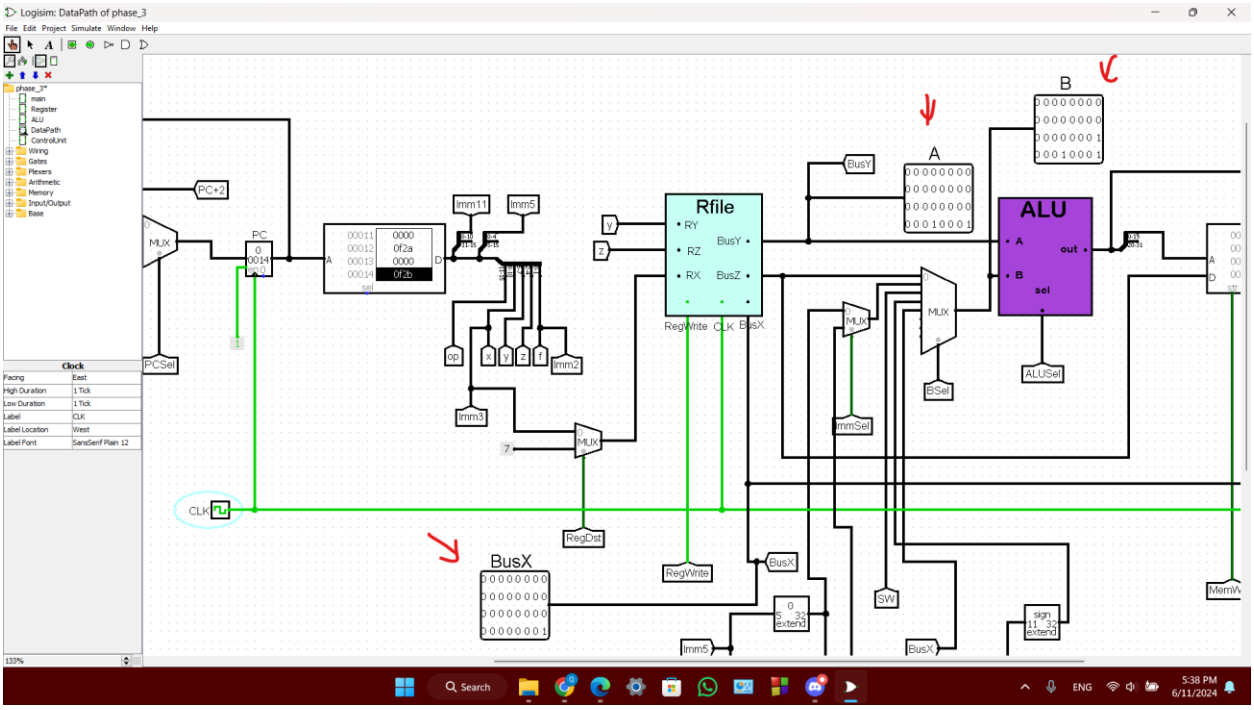

Figure 23 SLT instruction

10-ANDI

The instruction is 0x1730

ANDI R7,R1,2_10000

R1=0x00000011

The result will be 0000 0000 0000 0000 0000 0000 0010 0000

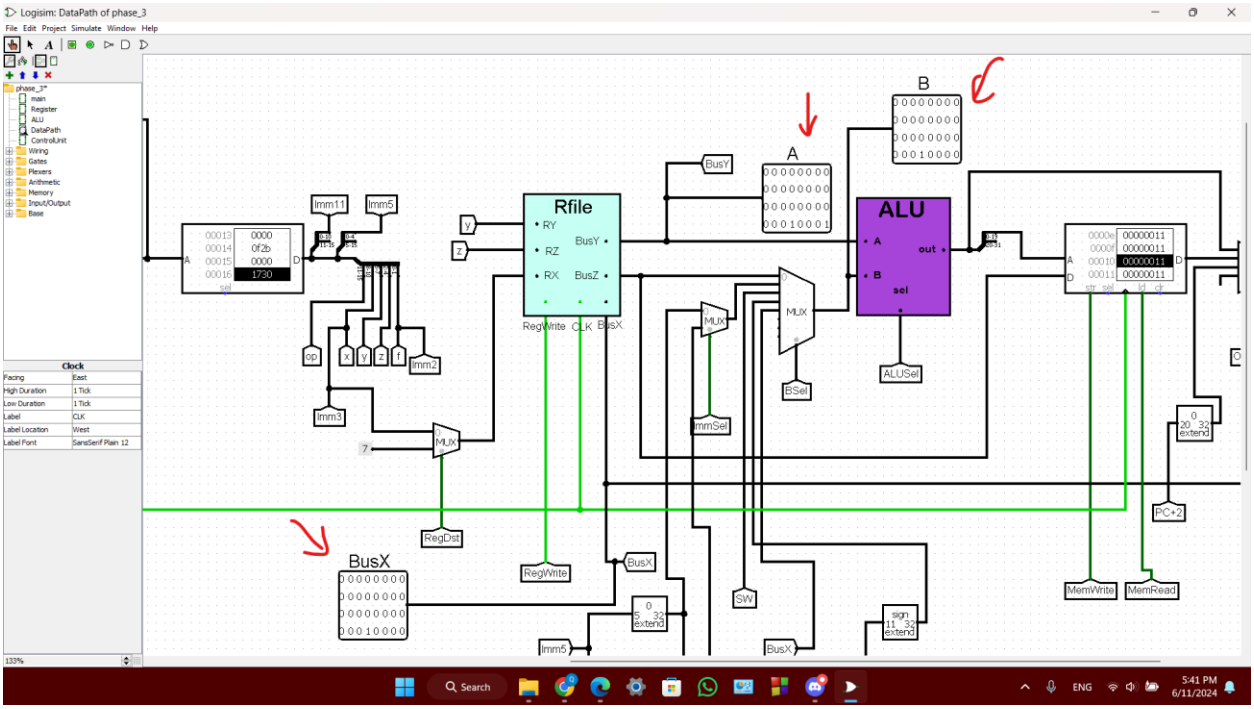

Figure 24 ANDI instruction

11-CANDI

0x1f30 is the instruction to be executed.
CANDI R7,R1,2_100000
R7=~R1 & immediate
Which was shown below in Bus X.

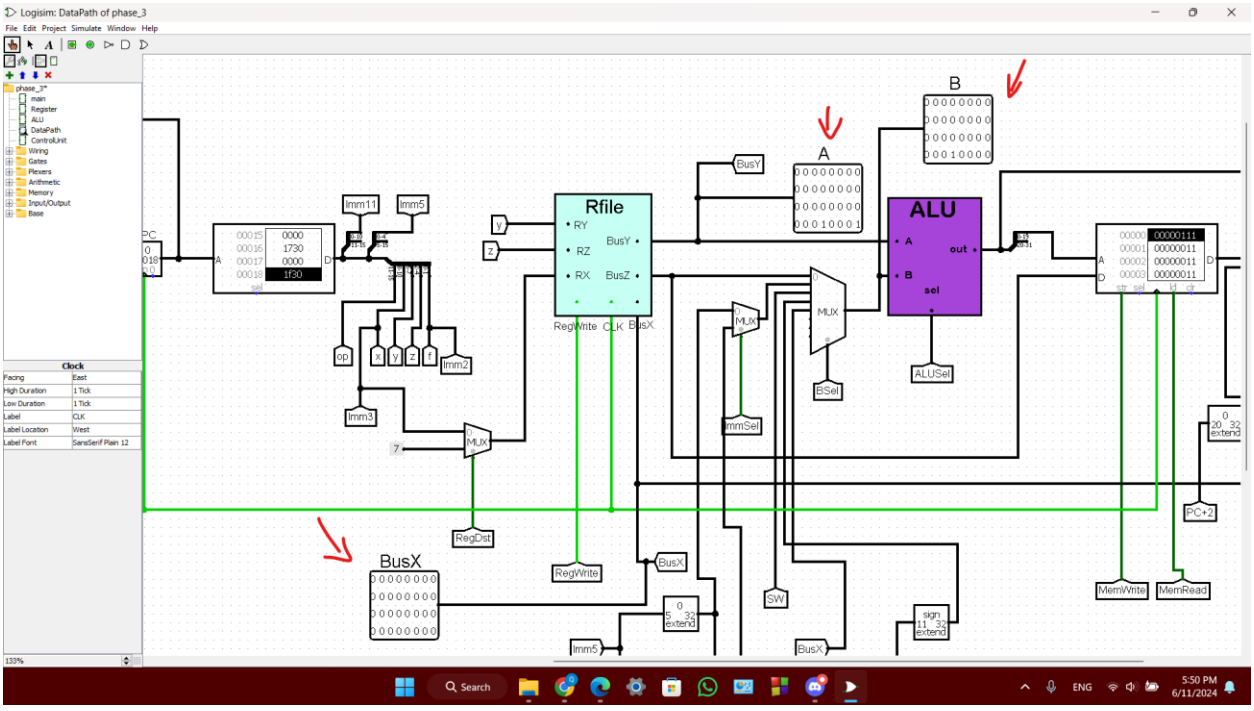

Figure 25 CANDI instruction

12-ORI

Instruction: 0x273F

ORI R7, R1, 2_11111

Result was true as shown below:

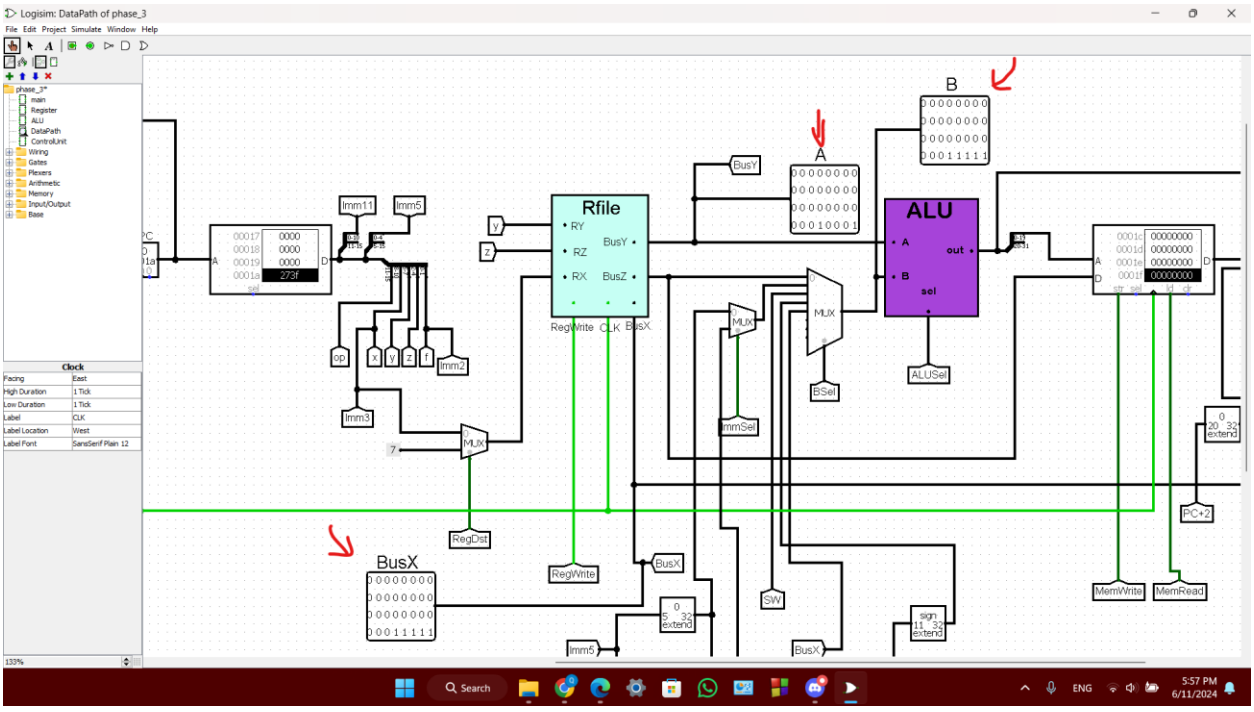

Figure 26 ORI instruction

13-XORI

0x2F3F was the instruction,

XORI R7, R1, 2_11111

The result was shown below:

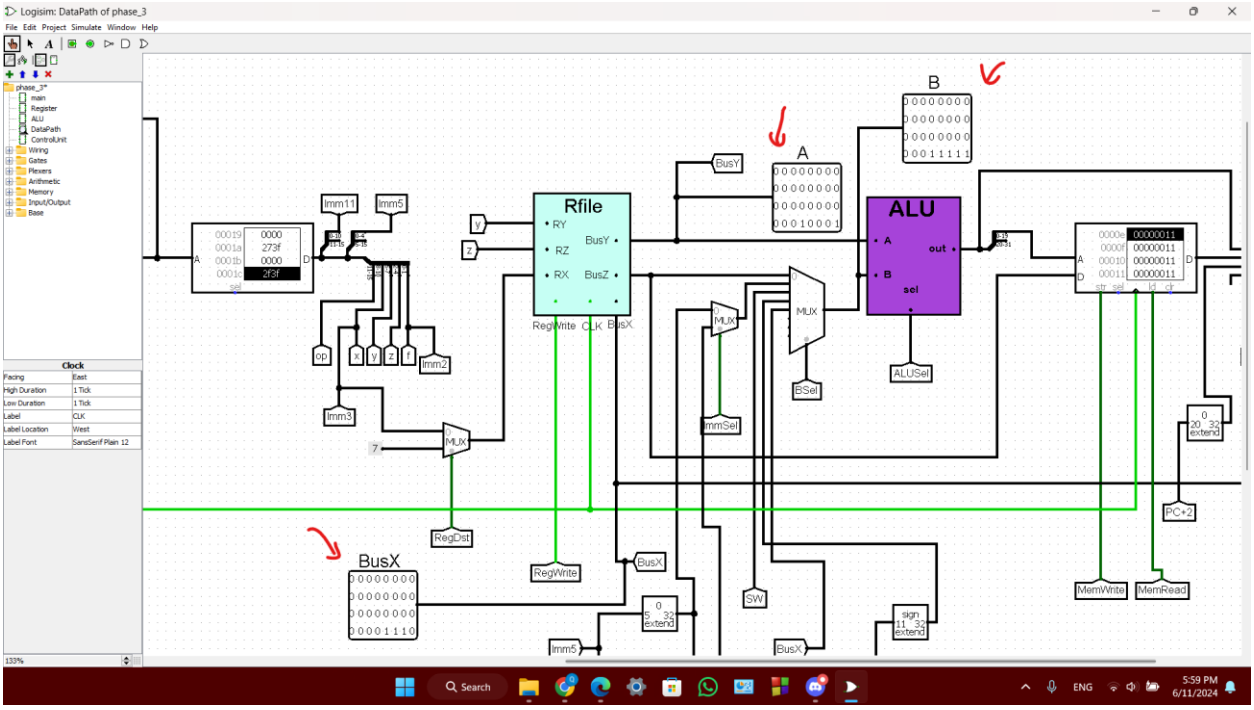

Figure 27 XORI instruction

14-ADDI

0x3721 is the instruction.

ADDI R7, R1, 2_00001

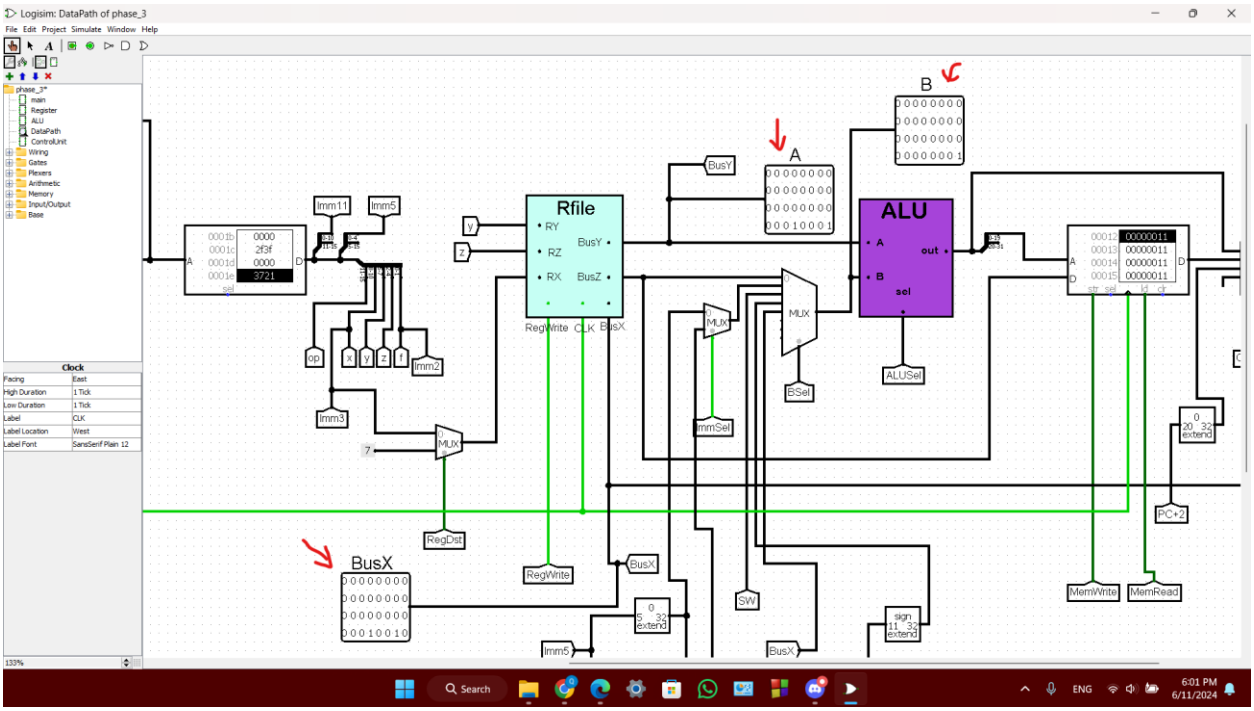

Figure 28 ADDI instruction

15-NADDI

The instruction is 0x3F3F

NADDI R7, R1 ,2_11111

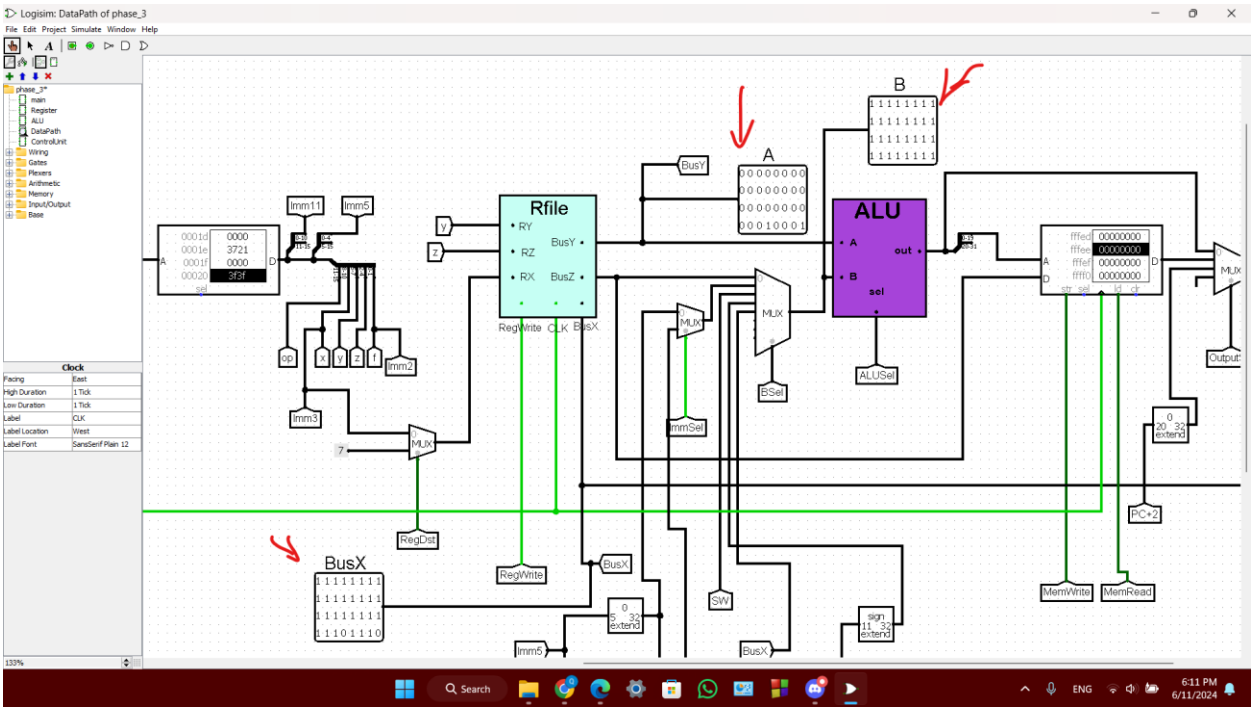

Figure 29 NADDI instruction

16-SEQI

Instruction: 0x472E

SEQI R7, R1, 2_01110

Since they are not equal, the result will be zero as shown below in BusX.

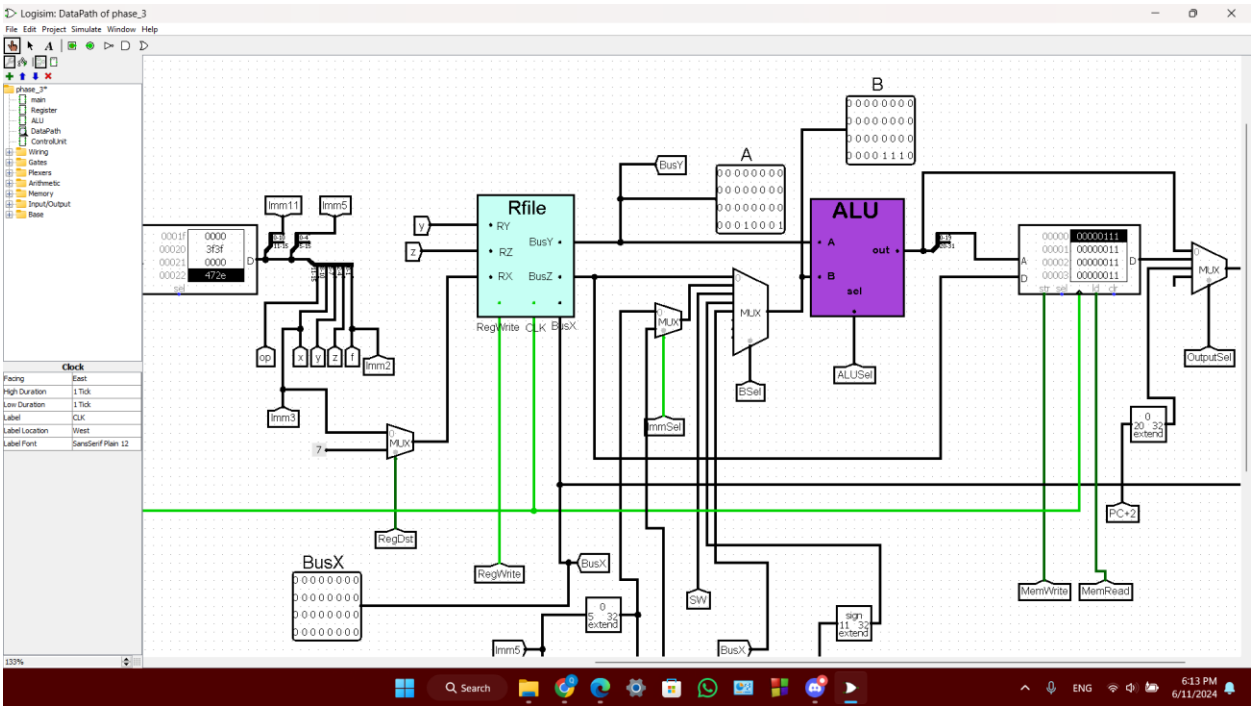

Figure 30 SEQI instruction

17- SLTI

0x4E2A is the instruction.

Since R1 > 2_01010 the result will be zero.

SLTI R6, R1, 2_01010

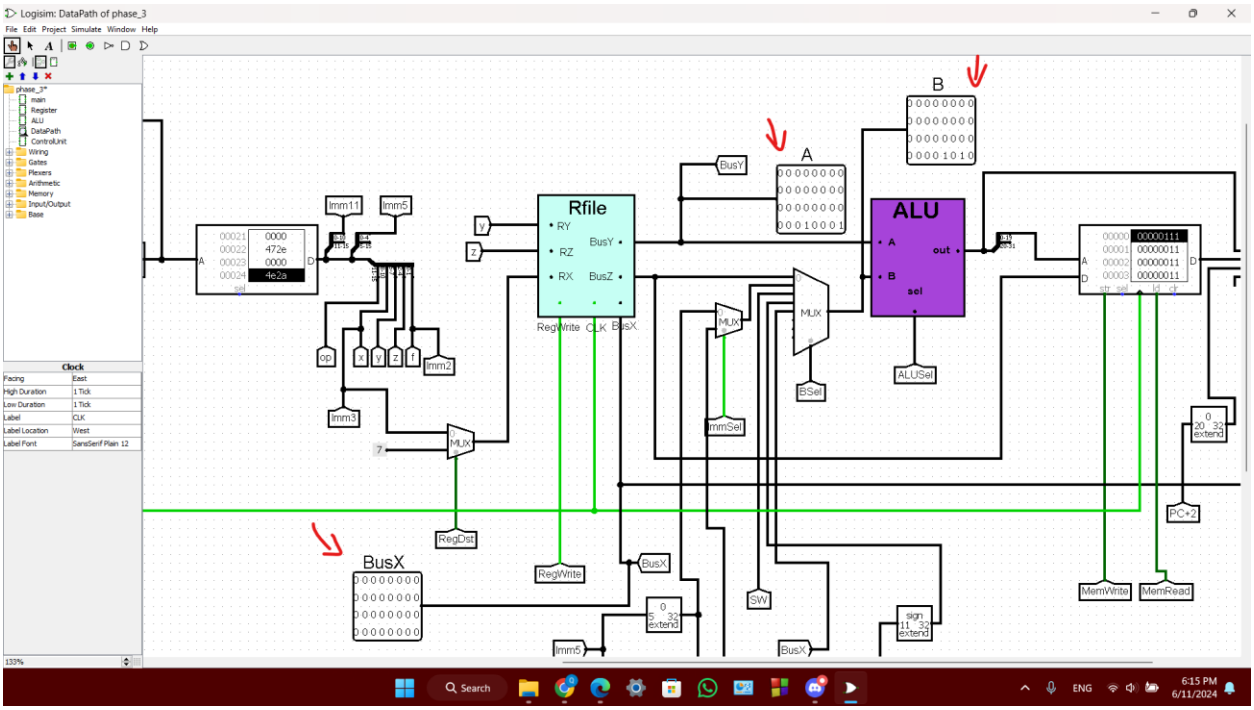

Figure 31 SLTI instruction

18- SLL

Instruction: 0x5721

SRL R7, R1, 2_00001

The result was shifted to the left by one.

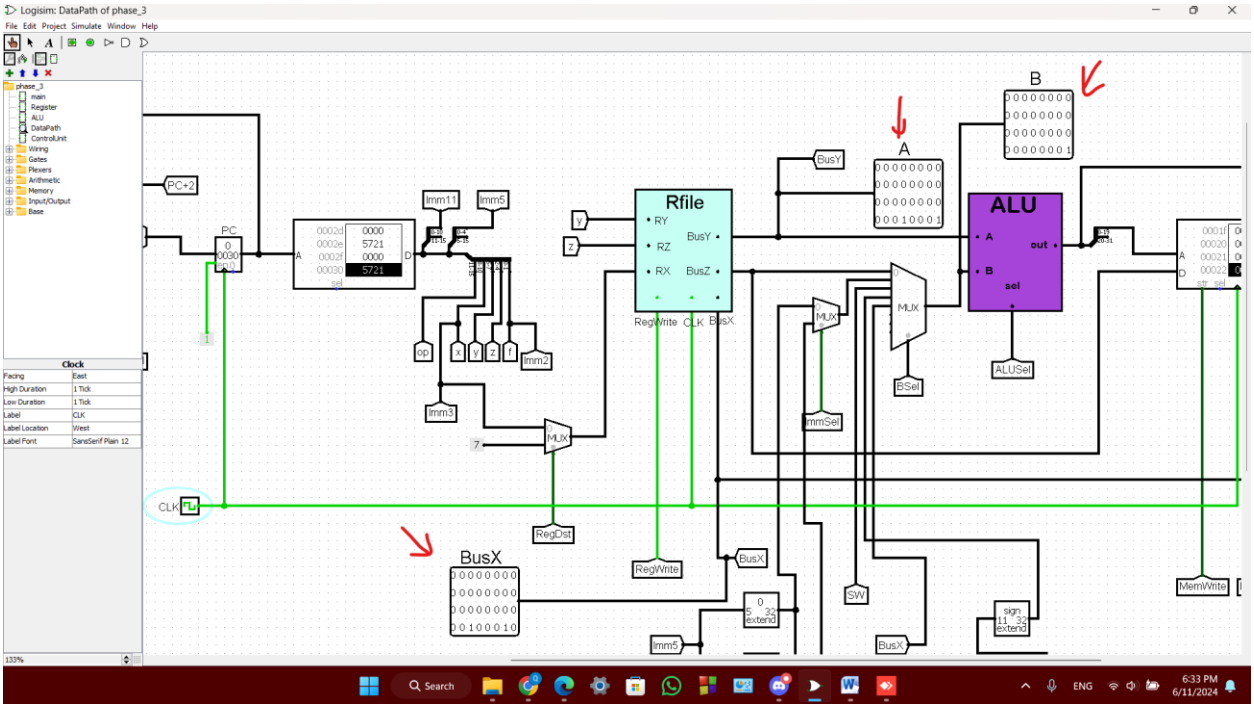

Figure 32 SLL instruction

19-SRL

Instruction is 0x5F22

SRL R7, R1, 2_00010

The result was shifted to the right by two.

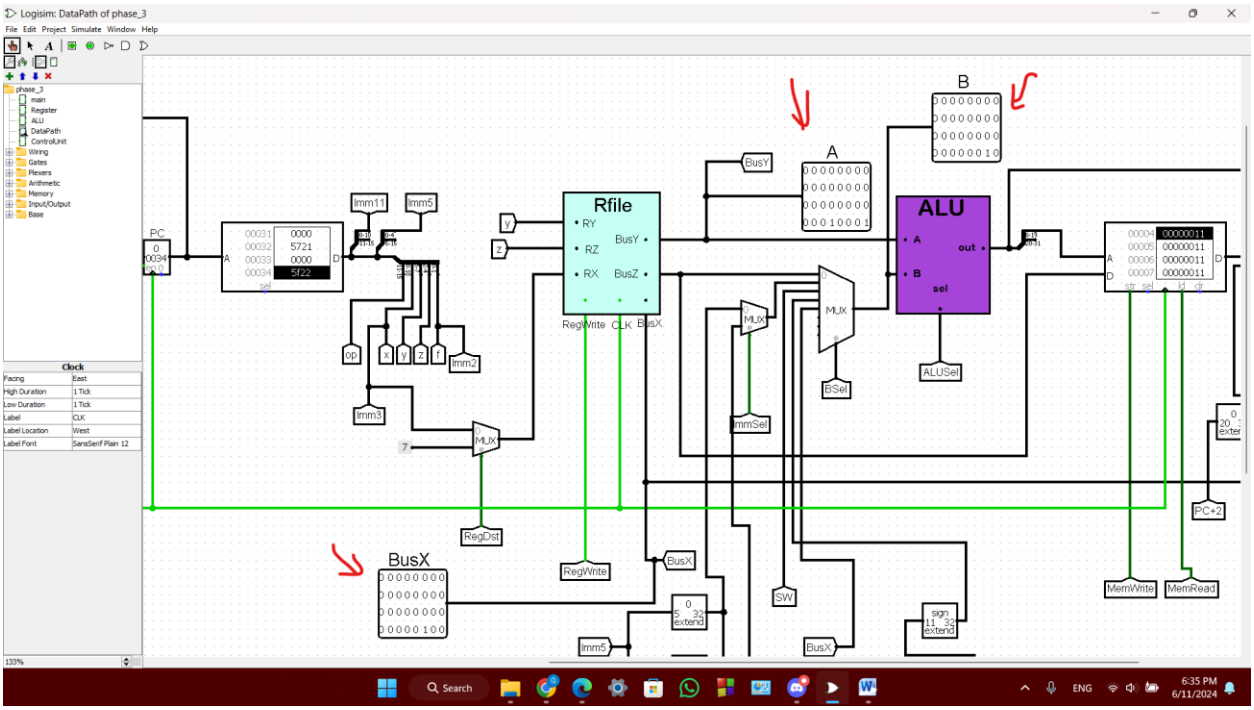

Figure 33 SRL instruction

20-SRA

Instruction: 0x6724

SRA R7, R1, 2_00100

The result has been shifted to the right arithmetically by four.

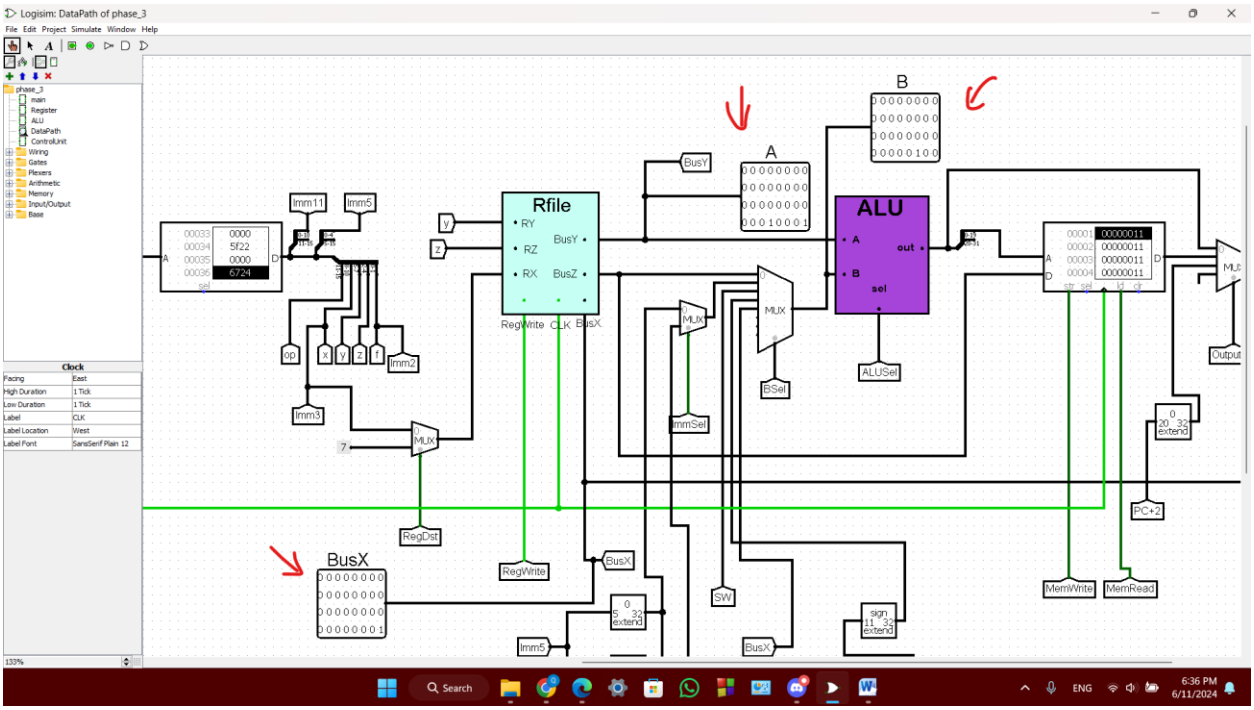

Figure 34 SRA instruction

21-ROR

Instruction: 0x6F22

ROR R7, R1, 2_00010

The result was rotated to right by 2.

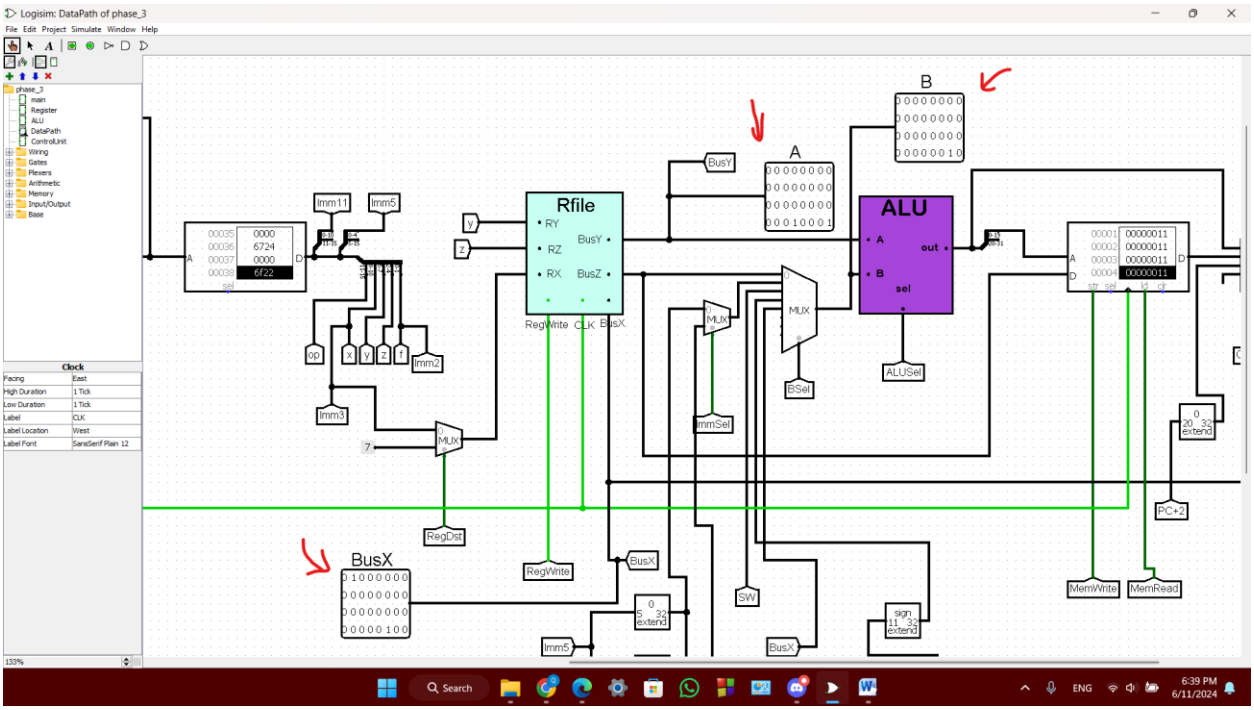

Figure 35 ROR instruction

22- SW

Instruction: 0x9B29

SW 2_000, R1, R2, 2_01

This instruction stores the value of R2 on memory location (R1+2_00001), where R1=2_010001

Since R2=0x00000111, the Memory location 0x00012 will be 0x00000111.

The memory was shown below:

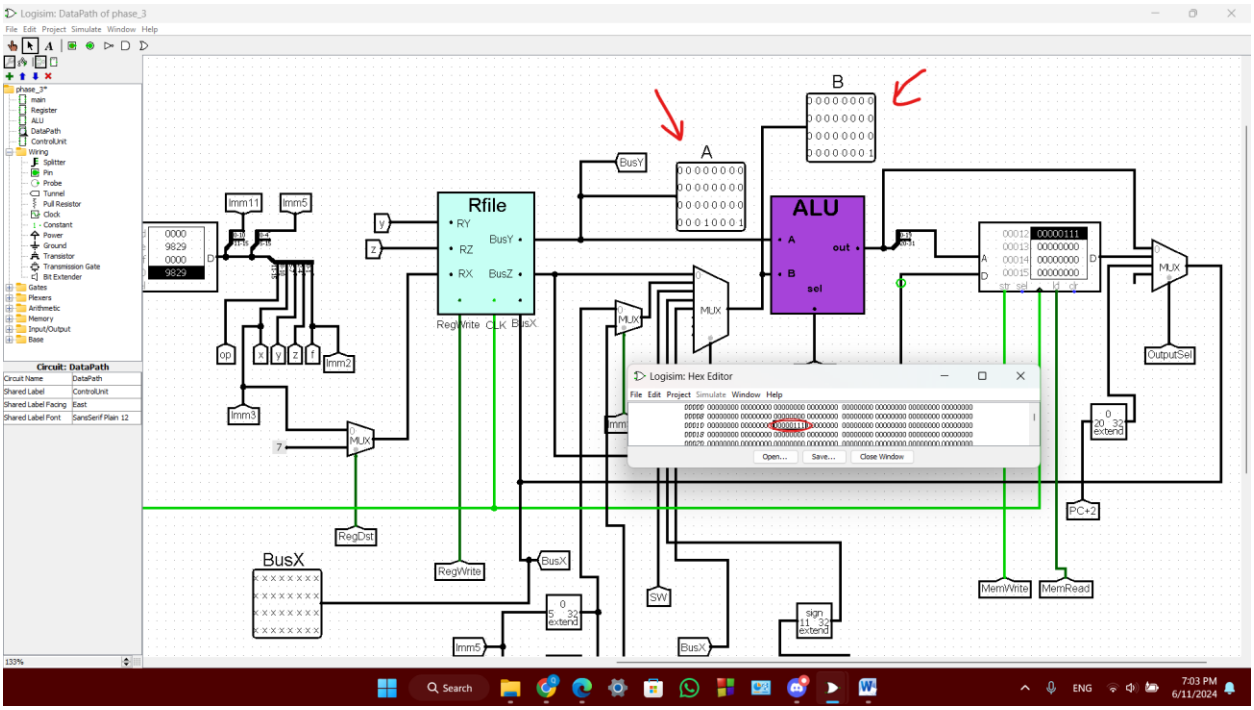

Figure 36 SW instruction

23- BEQ

We have load new values on R1,R2,R3:

R1=0x0000000A

R2=0x00000006=R3

Case 1 of BEQ:

Instruction: 0x7143

BEQ R1, R2, 2_00011

Since they are not equal, the ALU result will be zero, and the pc will increment by 2.

PC before Branch = 0x00008

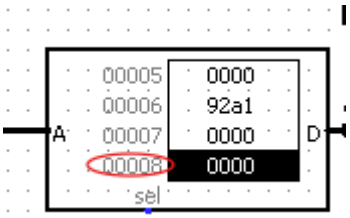

Figure 37 BEQ PC 1 before

PC after Branch = 0x0000a

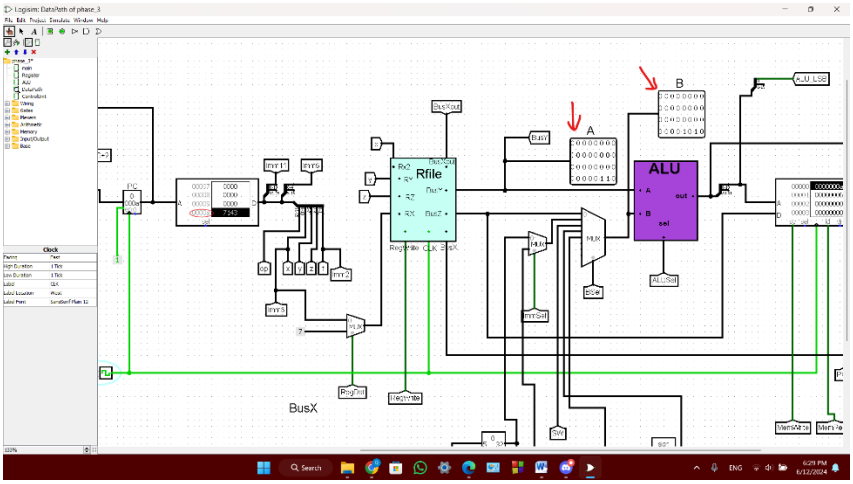

Figure 38 BEQ PC 1 after

Case2 of BEQ:

Instruction: 0x7263

BEQ R3, R2, 2_00011

Since they are equal, the ALU result was 1, and the increment of pc is: PC=PC + 6 (shift left of the immediate 3)

PC before:

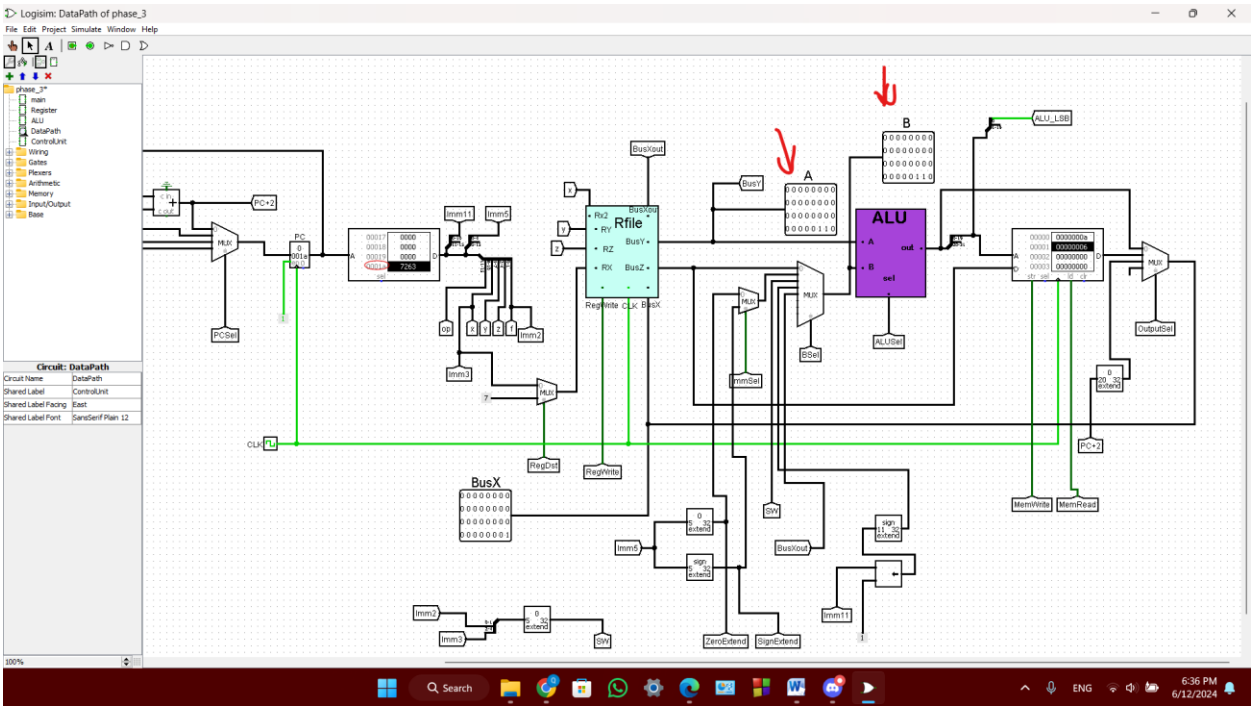

Figure 39 BEQ PC 2 before

PC after:

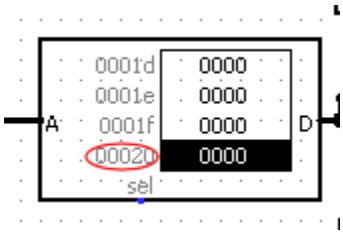

Figure 40 BEQ PC 2 after

24-BNE

Case 1 of BNE:

instruction 0x7A61

BNE R3,R2,2_00001

Since they are equal, the ALU result was 1, and the pc has been incremented by 2.

Before:

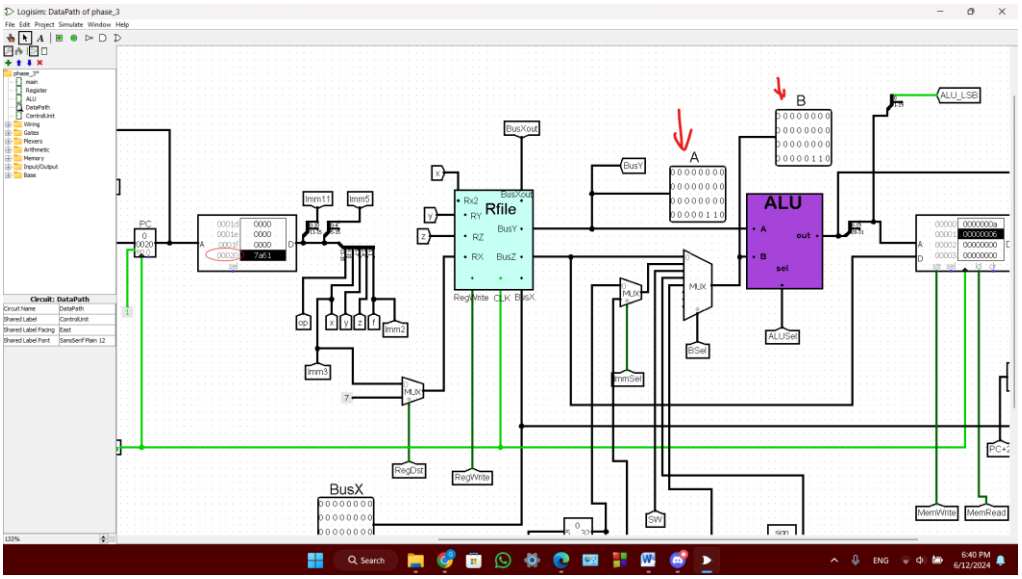

Figure 41 BNE PC 1 before

After:

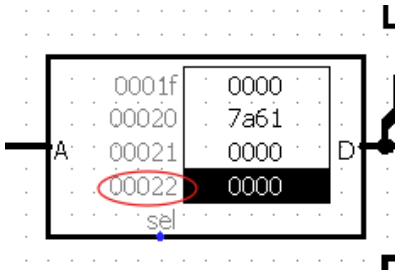

Figure 42 BNE PC 1 after

Case 2 of BNE:

instruction 0x7962

BNE R1,R3,2_00010

Since they are not equal -> PC=PC+4

Before:

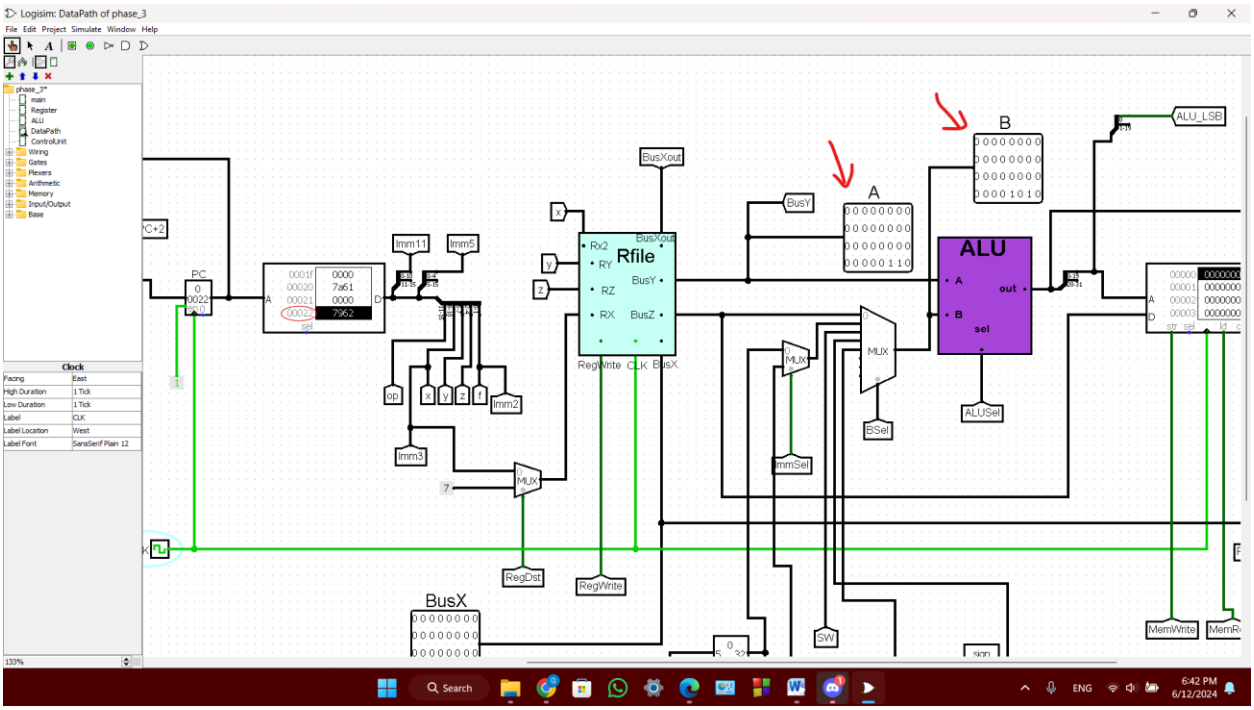

Figure 43 BNE PC 2 before

After:

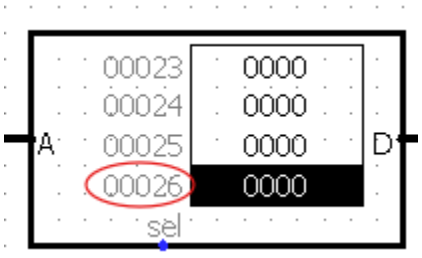

Figure 44 BNE PC 2 after

25-BLT

Case 1 of BLT:

instruction: 0x8222

BLT R2, R1, 2_00010

Before:

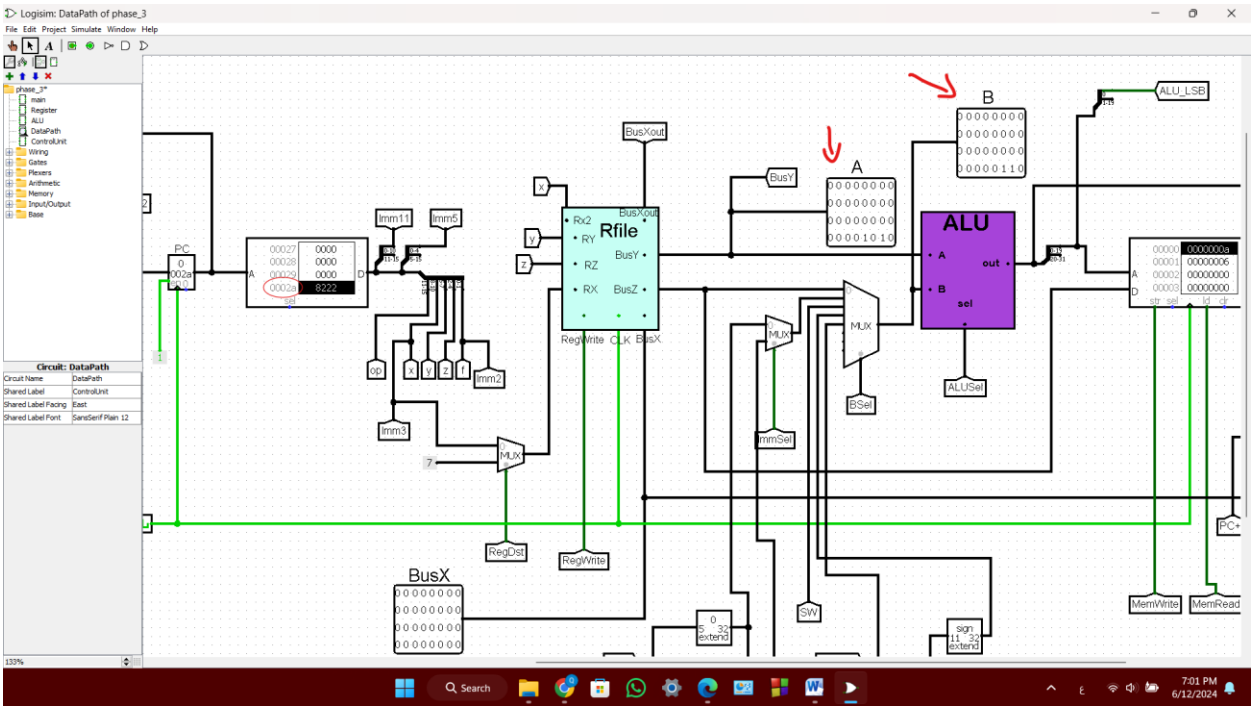

Figure 45 BLT PC 1 before

After: The pc has been incremented by 2.

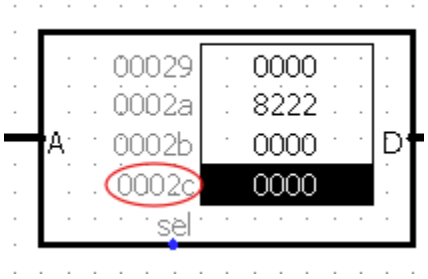

Figure 46 BLT PC 1 after

Case 2 of BLT:

instruction: 0x8142

BLT R1, R2, 2_00010

Before:

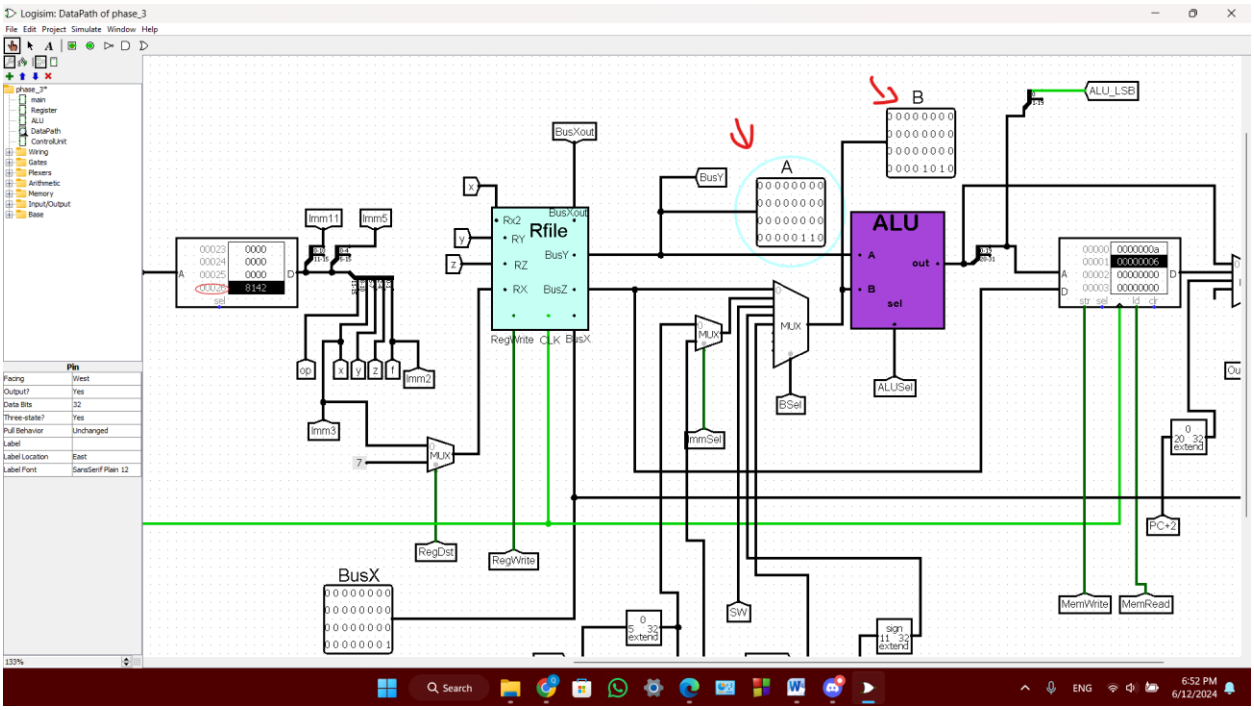

Figure 47 BLT PC 2 before

After: The pc has been incremented by 4.

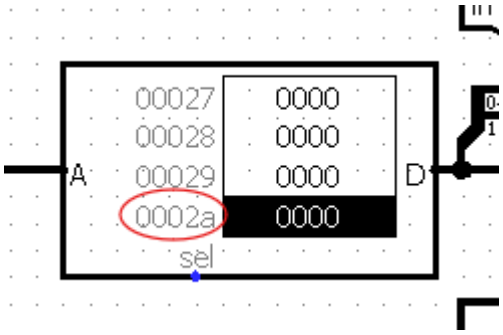

Figure 48 BLT PC 2 after

26-BGE

Case 1 of BGE:

Instruction: 0x8942

BGE R1, R2, 2_00010

Before:

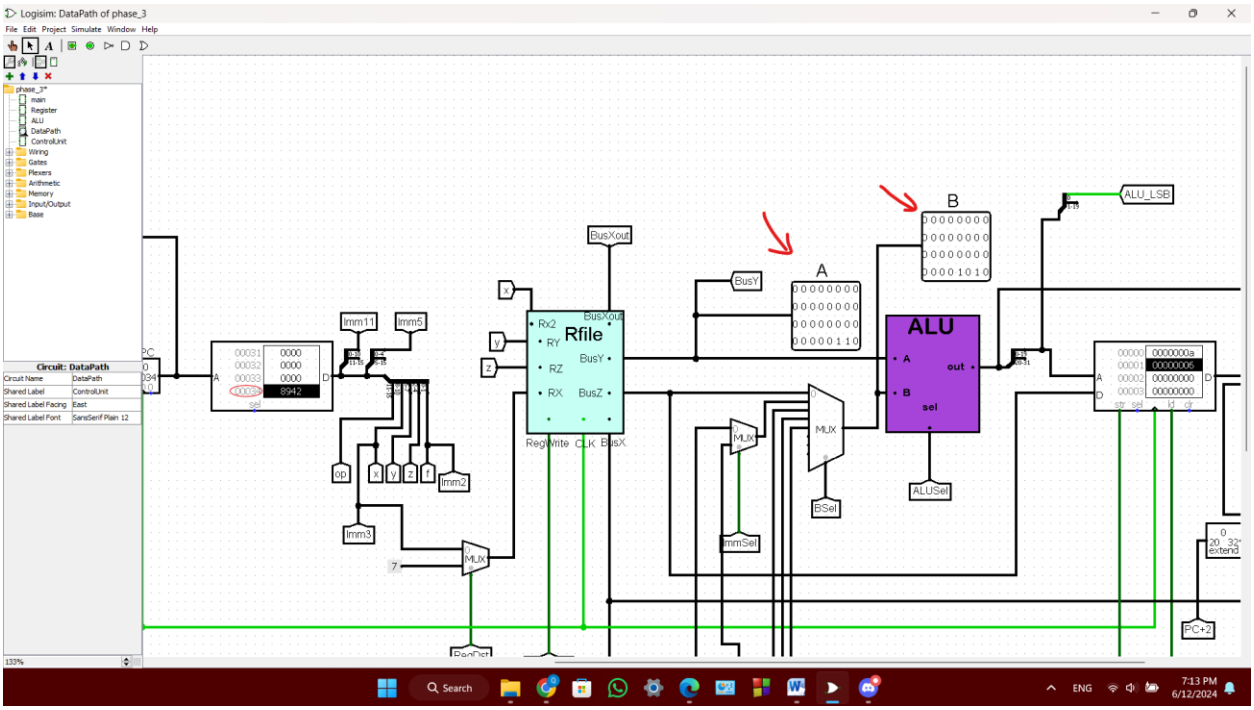

Figure 49 BGE PC 1 before

After: The pc has been incremented by 2.

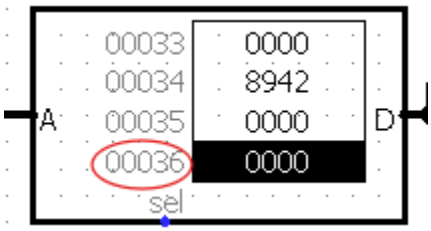

Figure 50 BGE PC 1 after

Case 2 of BGE:

Instruction: 0x8922

BGE R1, R2, 2_00010

Before:

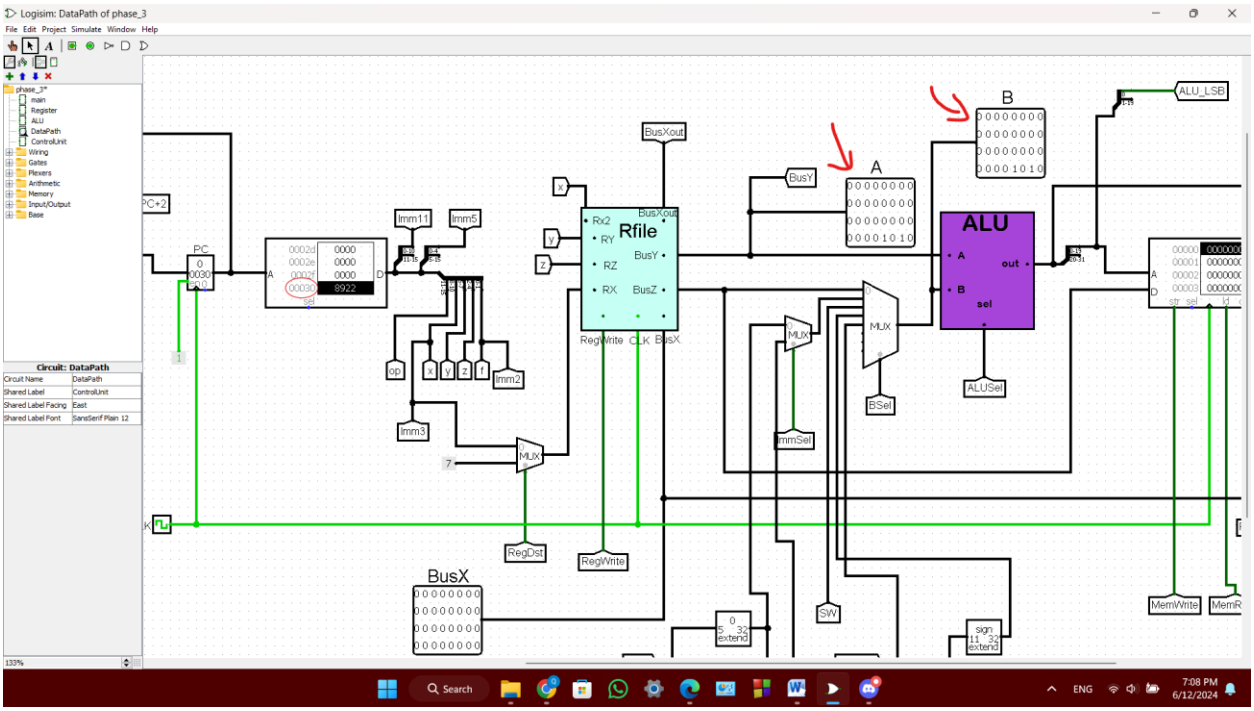

Figure 51 BGE PC 2 before

After:

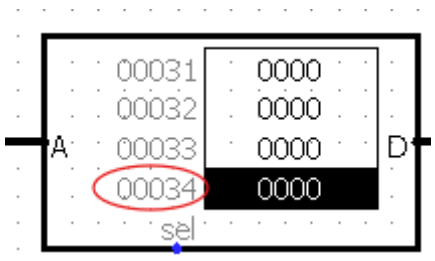

Figure 52 BGE PC 2 after

The pc has been incremented by 4.

27-J

Instruction: 0xA004

J 2_00100

PC before Jump:

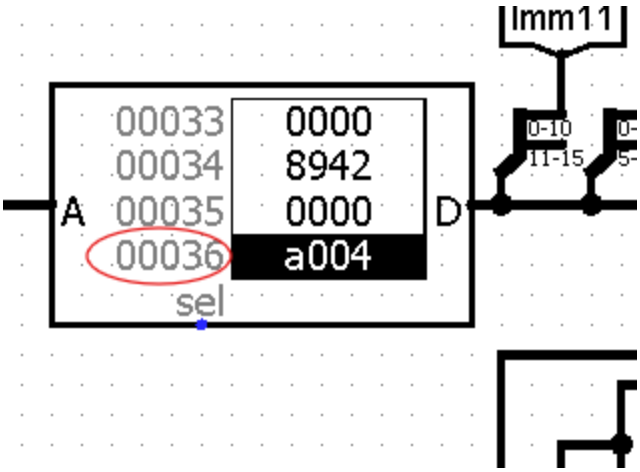

Figure 53 Jump PC before

After:

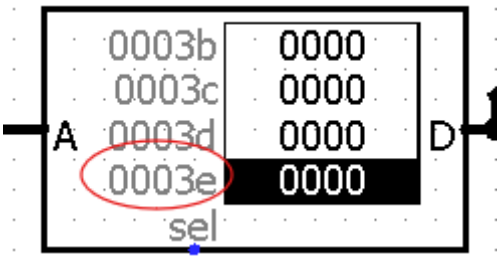

Figure 54 Jump PC after

PC has been incremented by 8.

28-JAL

Instruction: 0xA802

JAL 2_00010

The red wire means that the ALU result is neglected.

BusX shown in the figure below has the value of pc+2 [pc =0x0003e]

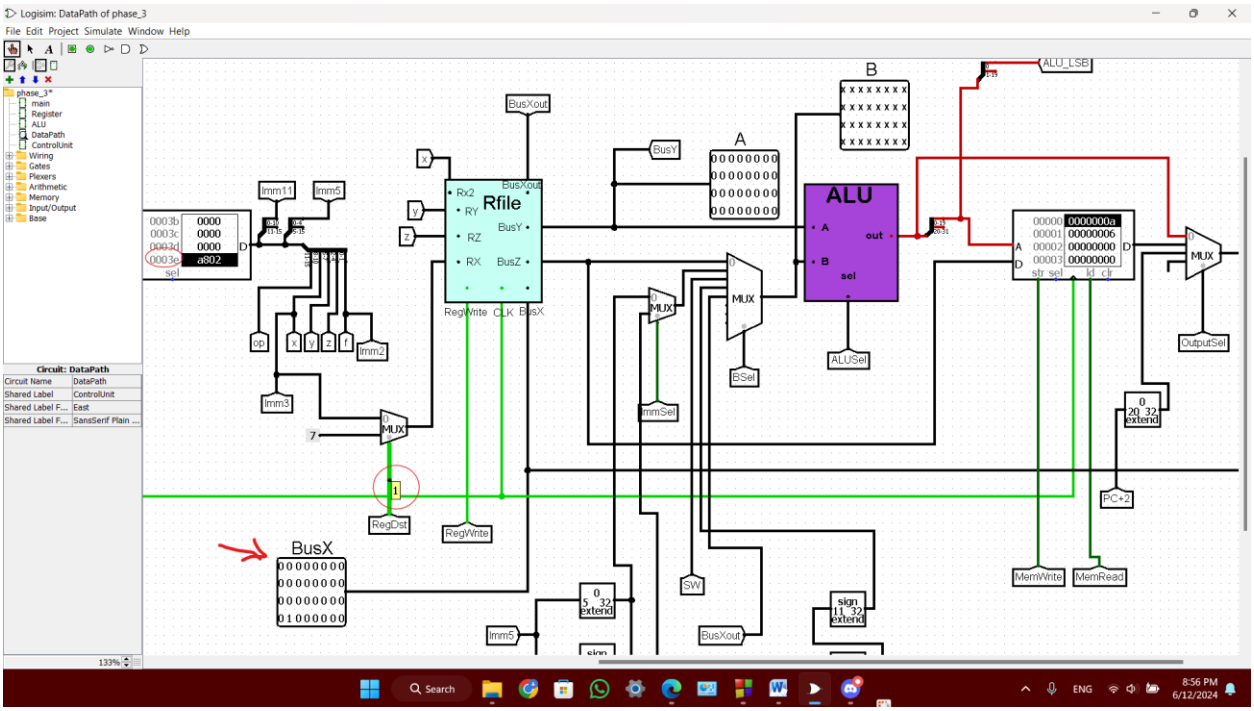

Figure 55 JAL instruction

Also, the PC new value has been incremented. So the new value of the pc is 0x00042.

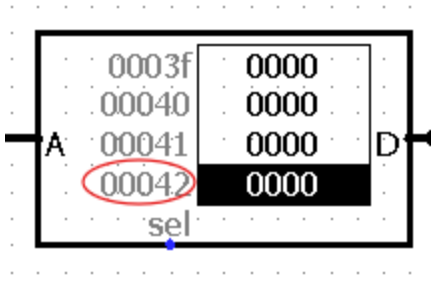

Figure 56 JAL PC

Simple code for Branch instruction

```
;all registers are initially zeros
ADDI R7, R7, 9   ; R7 = 9
ADDI R5, R5, 3  ; R5 = 3
ADD R6, R5, R7   ; R6 = R5 + R7 = 12
BEQ R6, R5, Loop ; If R6 == R5, branch to Loop
ADD R3, R6, R6   ; R3 = 0 if true-> skip
Loop: J Loop     ; Stay here forever
;1220204 Qasim
;1220006 Taleed
```

Design Alternatives, Issues and Limitations Part

All the instructions worked properly as shown in the simulation and testing part.

We have not found any alternative methods for our design and implementation because everything was as expected.

Team Work Part

- **Phase one:**

The register file is done by Taleed Hamadneh

The ALU is done by Qasim Batrawi

- **Phase two:**

Taleed Hamadneh has connected the PC with the ROM and the ROM with the RFile.

Qasim Batrawi has connected the RFile with the ALU and the ALU with the RAM.

Addressing modes for the PC is done by both Qasim Batrawi and Taleed Hamadneh.

- **Phase three:**

Connecting the control unit is done by Qasim Batrawi.

Inserting data to the OpCode ROM is done by Qasim Batrawi

Inserting data to the ALU ROM is done by Taleed Hamadneh

Inserting data to the PC ROM is done by Taleed Hamadneh

Testing is done by both Qasim and Taleed.

Report is done by both Taleed and Qasim.