Computer Organization and Microprocessor (ENCS2380)

Project: Single Cycle processor Design

Date: 11/6/2024

Prepared by:

Shaimaa dar taha 1222440 Section 2

Lina Ahmad 1220619 Section 1

Marwa Faqeeh 1220039 Section 1

Table of Contents:

Table of Figures:

Table of tables:

Designing a 32-bit processor with 16-bit instructions:

The architecture of a 32-bit processor with 16-bit instructions includes several components, such as:

Instruction Set Architecture (ISA):

Abstract interface between hardware and software in a computer system.
ISA defines the instruction set and the format in which instructions are encoded. It specifies how the processor interacts with memory, handles data, and executes operations.

Types of instructions in this project:

1) R-type:

5-bit Operation code (Op), 3-bit register numbers (x, y, and z), and 2-bit function field f

5-opcode	3-x	3-y	3-z	2-f
MSB				LSB

Figure 1:R-type instruction

2) I-type:

5-bit Op, 3-bit register numbers (x and y), and 5-bit Immediate

5-opcode	3-x	3-y	Imm5

Figure 2: I-type instruction

3) S-type:

5-bit Op, 3-bit register numbers (y and z), and 5-bit Immediate split into (Imm3 and Imm2)

5-opcode	Imm3	3-y	3-z	Imm2

Figure3: S-type instruction

4) J-type:

5-bit Op and 11-bit Immediate

5-opcode	Imm11

Figure4:J-type instruction

Register File:

* **Temporary Storage**: Registers in the register file hold operands for arithmetic and logic operations

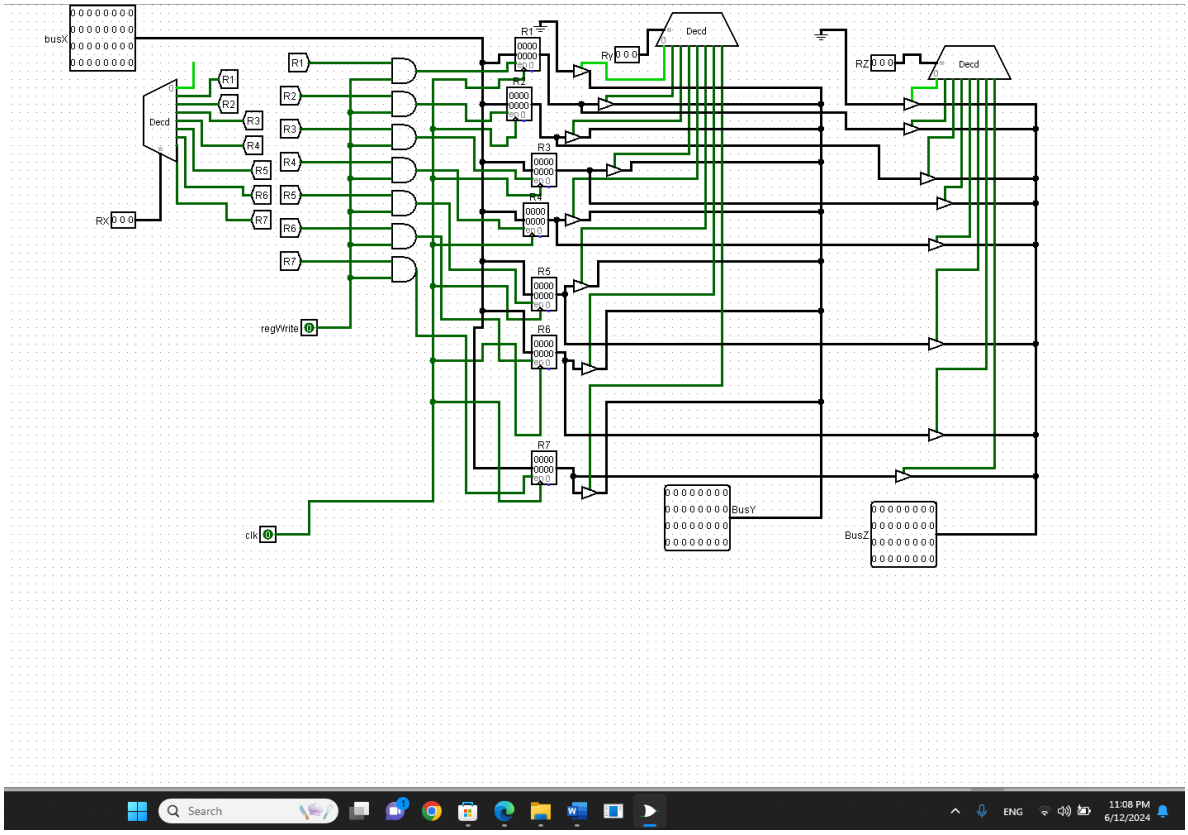


Figure5: register file

A RISC processor that has seven 32-bit registers (i.e. R1 to R7). Register R0 is hardwired/connected to zero (The value written to R0 is neglected).

Register	Number	Definition	Function
Register x (**Rx**)	Destination register	**Rx** is the name and value of register **x**	Always written and never read
Register y (**Ry**)	First source register	**Ry** is the name and value of register **y**	Always read and never written
Register z (**Rz**)	Second source register	**Rz** is the name and value of register **z**	Always read and never written

Figure 6: registers in our project

How to network Register File and selection registers (i.e. R1 to R7):

We used the decoder. We have 8 registers, so we need 3 bits to identify the register(R0-R7)

We use the register to determine the desired register (Rx), and the value of the register is

changed to determine which register we want. For Example, if the value of Rx=101 and reg

write is equal 1, a record R5, and so on. (Rx =101 to write on R5).

if regWrite(control signals) equal 0 then NO record is selected even if the Rx contains a value.

Clk: to apply what we did, we put the clock on the register(R0-R7).

bus x: gives value to register(R0-R7).

To choose what to read the Ry, we use a decoder such that if the value of the Ry is equal 011, it chooses the R3 and transfers the content of the R3 to the Bus y, and so on. This is done by:

Controlled Buffer:

- The Controlled Buffer has two main inputs: data input and control input.
- The data input is where the data you want to control the flow of passes through.
- The control input is where you connect the control signal that determines whether the buffer allows data to pass through or not.
- The Controlled Buffer has one output, which is connected to the destination where you want the data to go.

To choose what to read the Rz, we use a decoder such that if the value of the Rz is equal 001, it chooses the R1 and transfers the content of the R1 to the Bus z, and so on.(as similar Ry).

Arithmetic and Logic Unit (ALU):

The Arithmetic and Logic Unit (ALU) is a component of the Central Processing Unit (CPU). It┐ is to perform arithmetic and logical operations

We want to develop a 32-bit ALU to handle all the required operations: XOR, AND, OR, CAND, ADD, SEQ, NADD, SLT, SRA, SRL, SLL, and ROR.

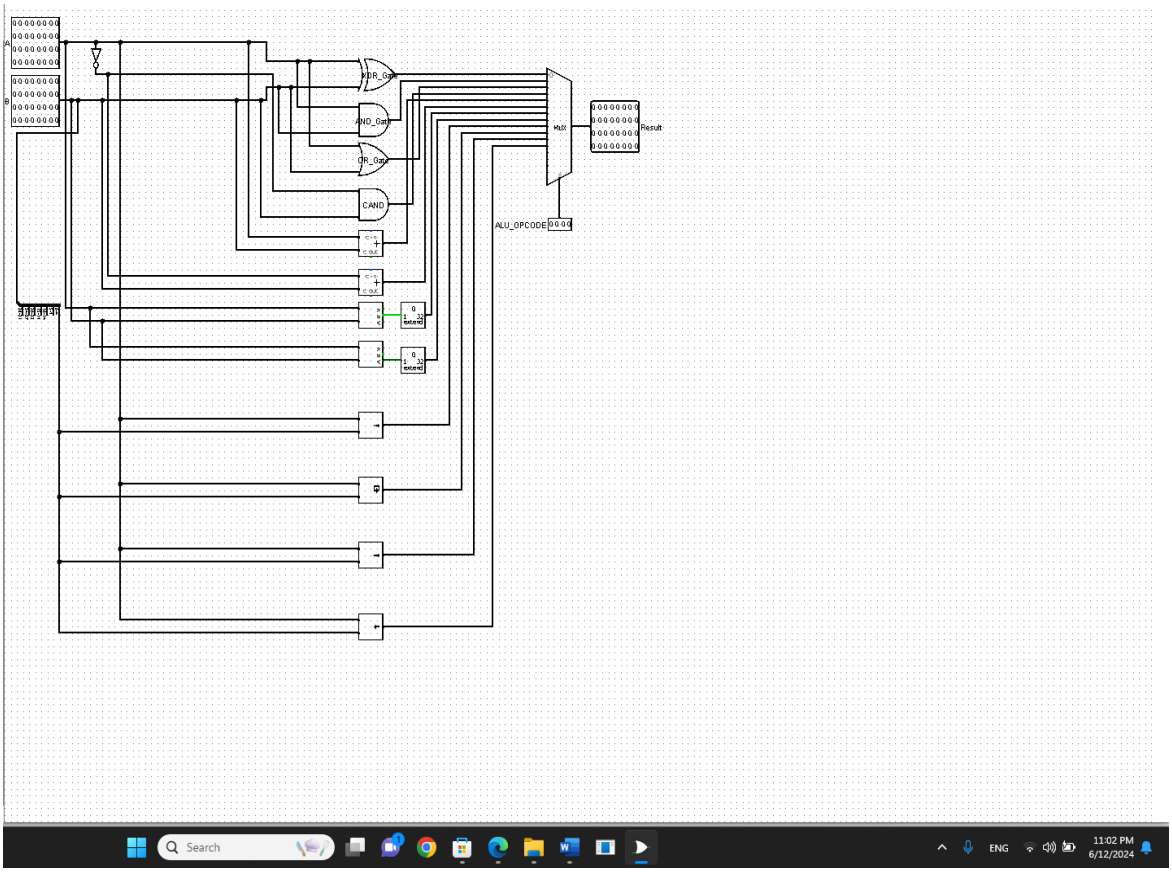


Figure7: :ALU

First we bring two inputs A and B which both have 32-bits to be the inputs for our ALU design, then we bring a MUX that have 32 data bits with 4 selection bits (because we have 12 operations to be connected to it so we need 4 selection bits which will be the opcode for the ALU).

We name it as ALU_OPCODE and it specifies the operation to be performed, then we bring the gates we mentioned above and set them all to 32 data bit to be connected to the inputs A and B in right way and the output of each of them connected to the MUX To be performed using ALU_OPCODE.

The instruction for each gate as below: The instruction for each gate as below:

Table1: instructions in our project

instruction	definition	op	x	y	z	f
AND	Rx = Ry & Rz	0	X	Y	Z	1
CAND	Rx = ~Ry & Rz	0	X	Y	Z	3
OR	Rx = Ry | Rz	0	X	Y	Z	2
XOR	Rx = Ry ^ Rz	0	X	Y	Z	0
ADD	Rx = Ry + Rz	1	X	Y	Z	0
NADD	Rx = –Ry + Rz	1	X	Y	Z	1
SEQ	Rx = (Ry == Rz) (result is 0 or 1)	1	X	Y	Z	2
SLT	Rx = (Ry < Rz) (result is 0 or 1)	1	X	Y	Z	3
SLL	Rx =Shift_Left_Logical(Ry, Imm5)	10	X	Y	Z	Imm5
SRA	Shift_Right_Arithmetic(Ry, Imm5)	11	X	Y	Z	Imm5
SRL	Rx = Shift_Right_Logical(Ry, Imm5)	12	X	Y	Z	Imm5
ROR	Rx = Rotate_Right(Ry, Imm5)	13	X	Y	Z	Imm5

In the instruction ADD we bring an Adder to do the job, also CADD but we add an inverter to the first input A

In the instruction SEQ we bring a comparator and we set it into 32bit data, we use zero extended to convert one bit to 32 bit to deal with the data bits in the MUX because comparator has one output and that makes an incompatible widths between the MUX and the comparator

same in the instruction SLT.

the instructions SLL, SRA, SRL which do the shift operations and ROR which do the rotate operation, we use a shifter to represent them.

the first input for each one was A and the second input was a five bit from input B(Imm5 (constant)) using splitter
(it's wrong to use a constant from the logisim directly we should do it at this way)
we set the bit width for splitter 32 bit and the fan out 7
Then we bring an output to the MUX named Result with 32bit data to represent the output of the ALU according to the ALU OPCODE

Format of instructions: we will try some instruction in our program

Table2 : format of instruction

instruction	format	binary	Hexa reb
AND R4 ,R3,R2	00000 100 011 010 01	0000 0100 0110 1000	**0X0468**
CAND R5, R3,R1	00000 101 011 001 11	0000 0101 0110 0101	**0X0565**
OR R3,R1,R2	00000 011 001 010 10	0000 0011 0010 1010	**0x032A**
XOR R7,R2,R1	00000 111 010 001 00	0000 0111 0100 0100	**0x0744**
ADD R3,R2,R1	00001 011 010 001 00	0000 1011 0100 0100	**0x0B44**
NADD R6,R3,R1	00001 110 011 001 01	0000 1110 0110 0101	**0x0E65**
SEQ R5,R3,R2	00001 101 011 010 10	0000 1101 0110 1010	**0x0D6A**
SLT R1,R3,R4	00001 001 011 100 11	0000 1001 0111 0011	**0x 0973**
ANDI R4,R2,5	00010 100 010 00101	0001 0100 0100 0101	**0x1445**
CANDI R5,R6,5	00011 101 110 00101	0001 1101 1100 0101	**0x1DC5**
ORI R4,R3,5	00100 100 011 00101	0010 0100 0110 0101	**0x2465**
XORI R5,R7,5	00101 101 111 00101	0010 1101 1110 0101	**0x2DE5**
ADDI R3,R1,5	00110 011 001 00101	0011 0011 0010 0101	**0x 3325**
NADDI R4,R5,5	00111 100 101 00101	0011 1100 1010 0101	**0x3CA5**
SEQI R4,R2,5	01000 100 010 00101	0100 0100 0100 0101	**0x2A88**
SLTI R3,R1,5	01001 011 001 00101	0100 1011 0010 0101	**0x4B25**
SLL R3,R1,5	01010 011 001 00101	0101 0011 0010 0101	**0x5325**
SRL R2,R5,5	01011 010 101 00101	0101 1010 1010 0101	**0x5AA5**
SRA R5,R6,5	01100 101 110 00101	0110 0101 1110 0101	**0x65C5**
ROR R1,R2,5	01101 001 010 00101	0110 1001 0100 0101	**0x6945**
BEQ R6,R4,5	01110 110 100 00101	0111 0110 1000 0101	**0x7685**
BNE R2,R3,5	01111 010 011 00101	0111 1010 0110 0101	**0x7A65**
BLT R5,R6,5	10000 101 110 00101	1000 0101 1100 0101	**0x85C5**
BGE R1,R2,5	10001 001 010 00101	1000 1001 0100 0101	**0x8945**
LW R6,R2,5	10010 110 010 00101	1001 0110 0100 0101	**0x9645**
SW 3,R3,R3,2	10011 011 011 011 10	1001 1011 0110 1110	**0x9B6E**
J 3,R2,R4,2	10100 011 010 100 10	1010 0011 0101 0010	**0xA352**
JAL 3,R1,R5,2	10101 011 001 101 10	1010 1011 0011 01 10	**0xAB36**

Op_code for Control Signals:

1) Pc cal

00>branch

01>pc+2

10>jump

2) dist

0>Rx

1>R7

3) RW

0>no write on reg

1>write on reg

4)ext

0>Zero ext. Imm5

1>sign ext. Imm5

5)ALUOP

0000>XOR//0001>AND//0010>OR//0011>CAND//0100>ADD

0101>NADD//0110>SEQ//0111>SLT//1000>SRA//1001>ROR

//1010>SRL//1011>SLL

6)b Sel

00>Bus z

01> Imm 5

10>Imm3 Imm 2

11>Imm 11

7)memory write

0>no write

1>write

8)memory read

0>no read

1>read

9)Wbdata

00> ALU result

01> Data Mem

10>Pc+2

Expected values for Control Signals:

instruction	F	opcde	Pc Cal	dist	Rw	ext	B sel	Memory write	Memory Read	Wb data	ALU OP	Hex
ADDI	xx	00110	01	0	1	1	01	0	0	00	0100	0x032D04
AND	00	00000	01	0	1	0	00	0	0	00	0001	0x002801
CAND	01	00000	01	0	1	0	00	0	0	00	0011	0x102803
OR	01	00000	01	0	1	0	00	0	0	00	0010	0x402802
XOR	00	00011	01	0	1	0	00	0	0	00	0000	0x002802
ADD	00	00001	01	0	1	0	00	0	0	00	0100	Ox00A804
NADD	01	00001	01	0	1	0	00	0	0	00	0101	0x10A805
SEQ	10	00001	01	0	1	0	00	0	0	00	0110	0x20A806
SLT	11	00001	01	0	1	0	00	0	0	00	0111	0x3A807
ANDI	xx	00010	01	0	1	0	01	0	0	00	0001	0x012901
CANDI	xx	00011	01	0	1	0	01	0	0	00	0011	0x01A903
ORI	xx	00100	01	0	1	0	01	0	0	00	0010	0x022902
XORI	xx	00101	01	0	1	0	01	0	0	00	0000	0x02A900
NADDI	xx	00111	01	0	1	1	01	0	0	00	0101	0x03AD05
SEQI	xx	01000	01	0	1	1	01	0	0	00	0110	0x042D06
SLTI	xx	01001	01	0	1	1	01	0	0	00	0111	0x04AD07
SLL	xx	01010	01	0	1	1	01	0	0	00	1011	0x052D0B
SRL	xx	01011	01	0	1	0	01	0	0	00	0101	0x05A905
SRA	xx	01100	01	0	1	0	01	0	0	00	1000	0x062908
ROR	xx	01101	01	0	1	0	01	0	0	00	1001	0x06A909
BEQ	xx	01110	01	0	0	1	01	0	0	00	0110	0x072506
BNE	xx	01111	00	0	0	1	01	0	0	00	0000	0x078500
BLT	xx	10000	00	0	0	1	01	0	0	00	xxxx	0x080500
BGE	xx	10001	00	0	0	1	01	0	0	00	xxxx	0x088500
LW	xx	10010	01	0	1	1	01	0	1	01	xxxx	0x092D10
SW	xx	10011	01	0	0	1	10	1	0	00	xxxx	0x09A680
J	xx	10100	10	0	0	0	00	0	0	00	xxxx	0x0A4000
JAL	xx	10101	10	1	1	1	11	0	0	10	xxxx	0x0ADF20

Table3: expected values for control signal

Control Unit:

The Control Unit orchestrating the execution of instructions received by the CPU. Its primary function is to interpret and execute instructions stored in memory. When the Control Unit fetches an instruction, it decodes the opcode (operation code) to determine the specific operation to be performed. Based on this opcode, the Control Unit generates control signals that direct the Arithmetic Logic Unit (ALU) to perform the required operation.

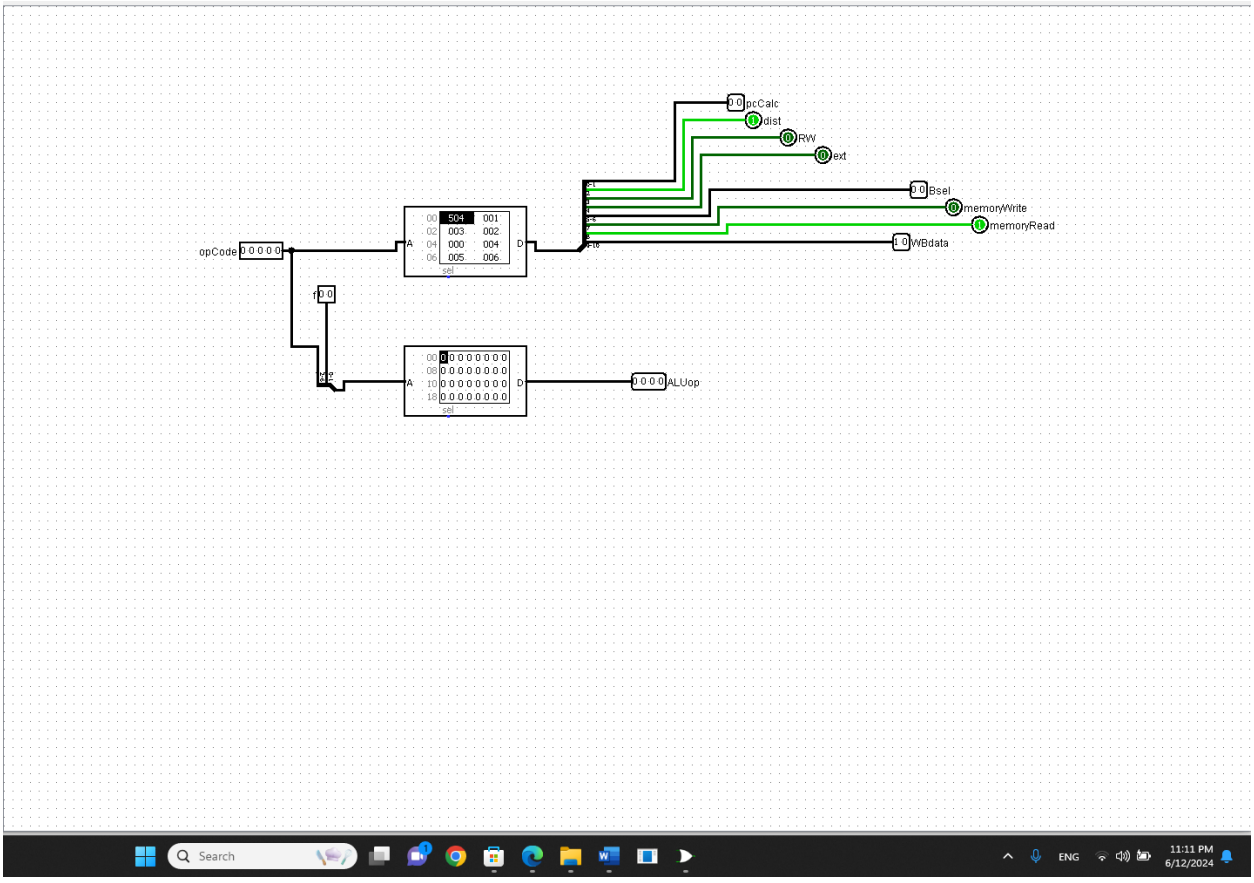


Figure8: *Control Unit*

Data Path:

The data path in a computer system is the pathway through which data flows, such as fetching, decoding, and executing instructions. It consists of registers, arithmetic logic units (ALUs), and other components interconnected by data buses, enabling the manipulation and processing of data as directed by the control unit.

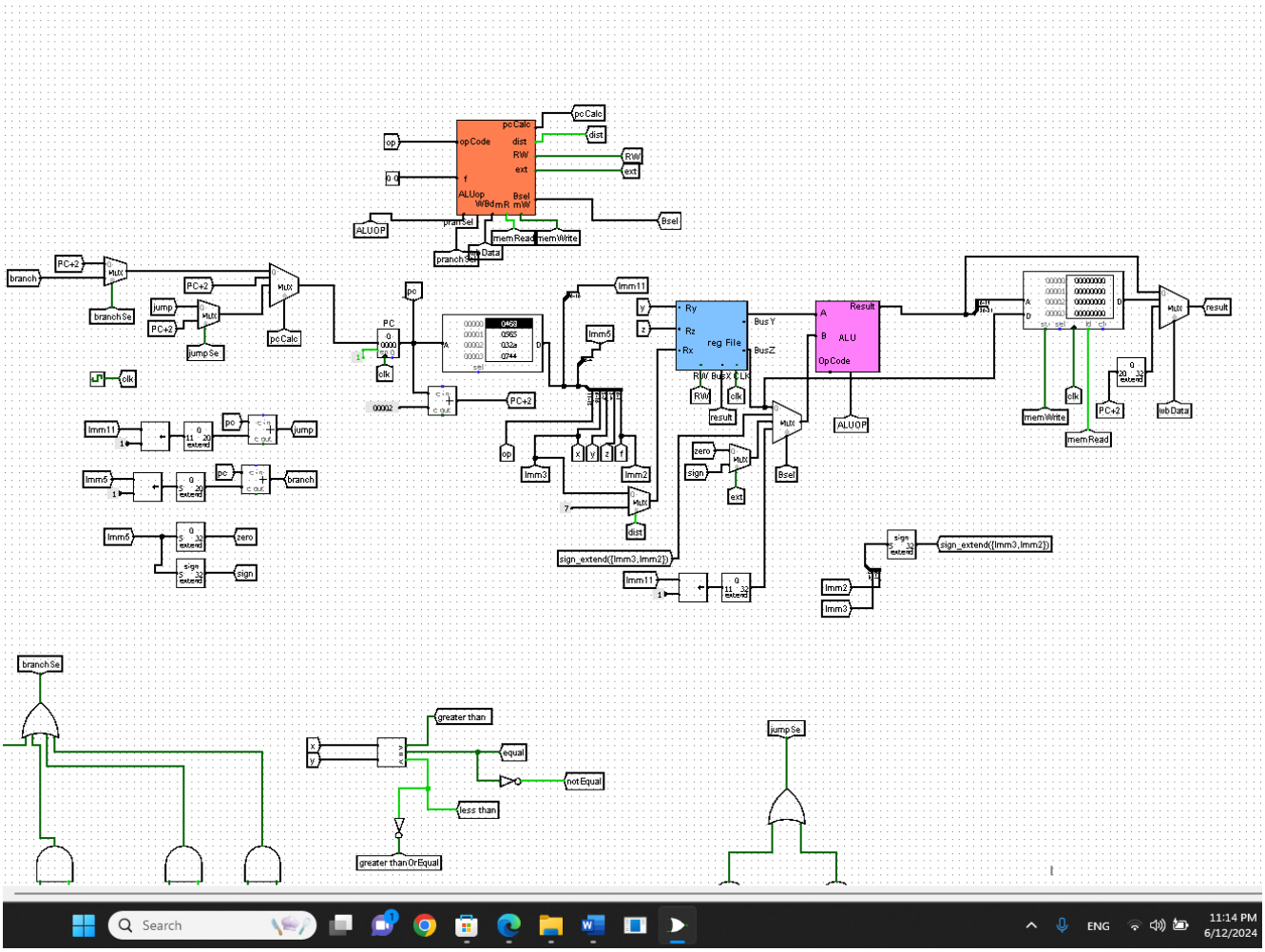


Figure 9:Data Path

Branch and jump :

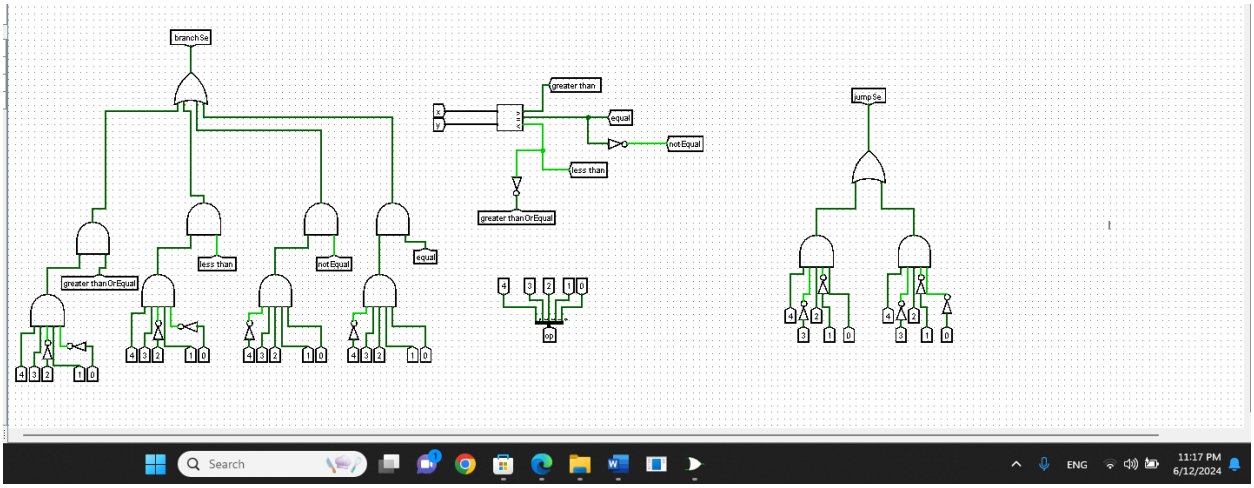
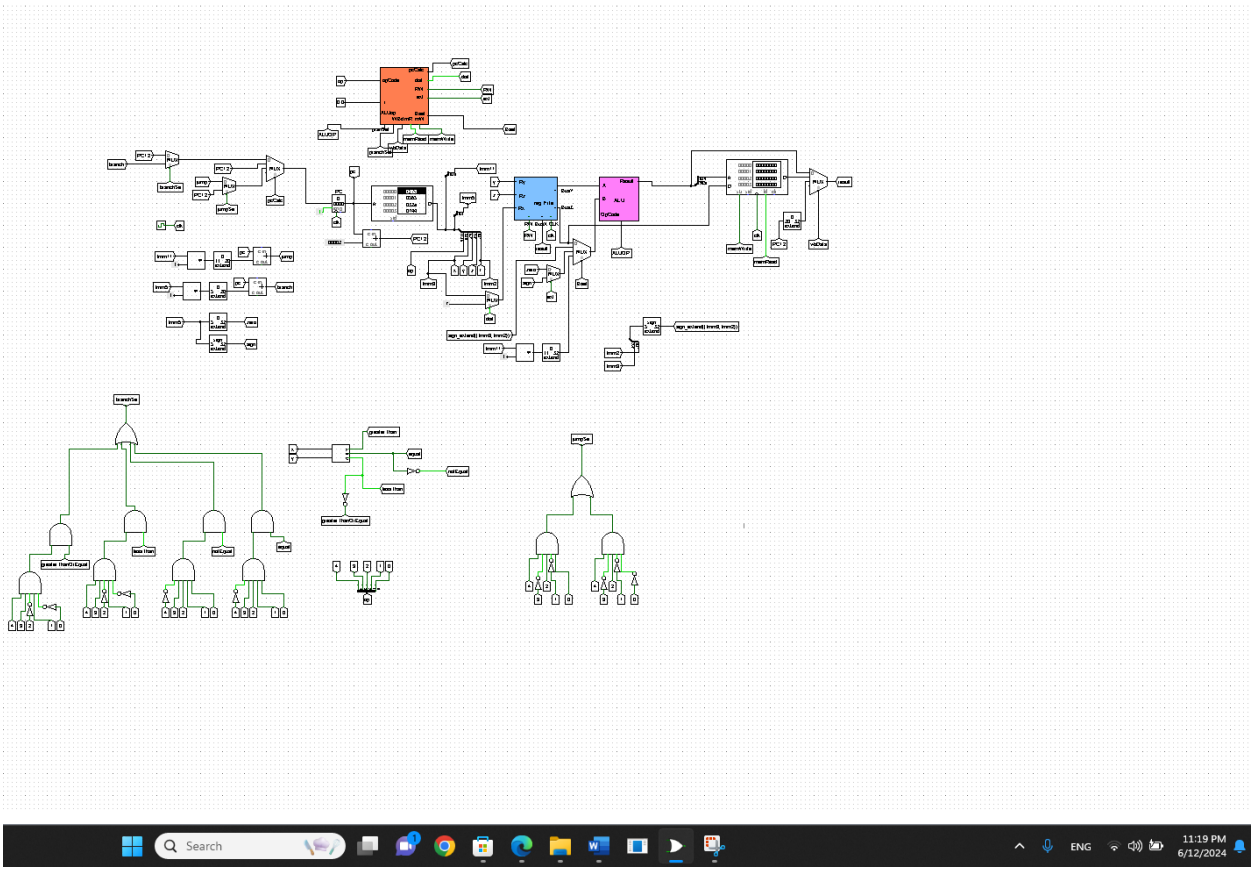


Figure10: Branch and jump

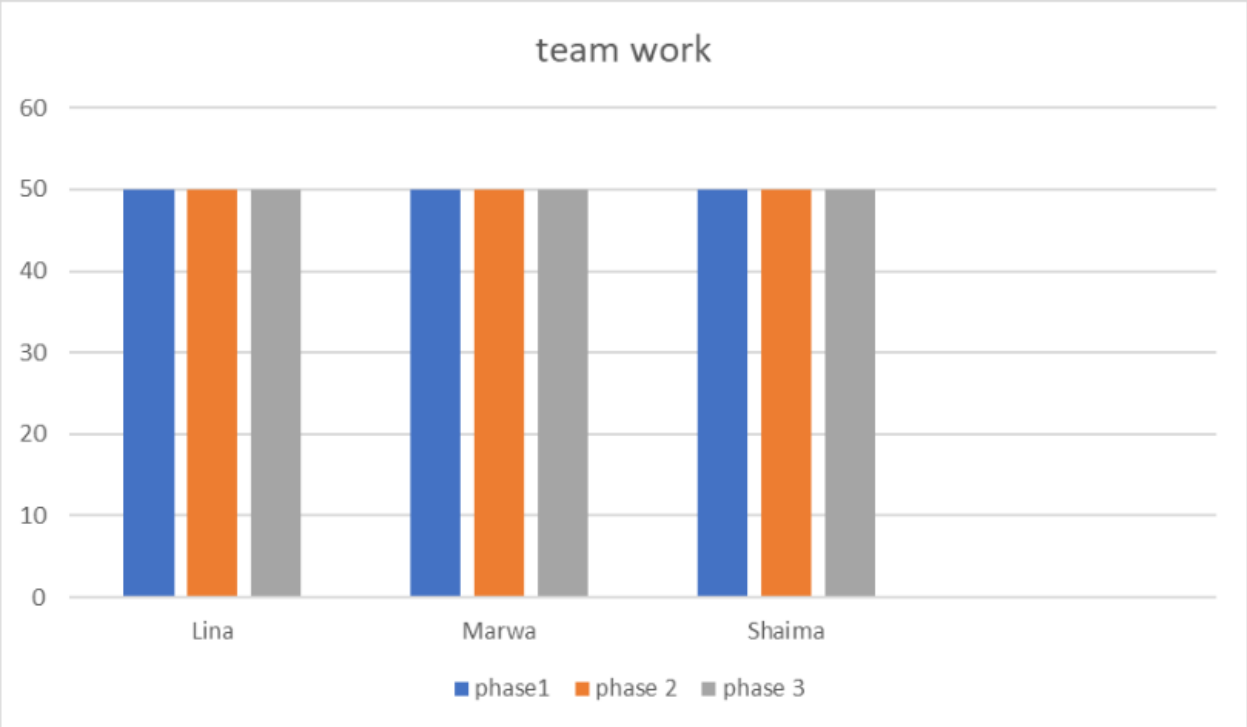


Codes:

* If (x1>x2) {f=y ^2 +2} else {f=y+1}.

Assembly:

```
LW R1, x1        # Load x1 into R1
LW R2, x2        # Load x2 into R2
LW R3, y         # Load y into R3
CMP R1, x2  ; Compare x1 and x2
BLE next ; Branch to next if x1 <= x2
MUL R4, R3, R3   ; Calculate y^2 (if x1 > x2)
ADD R4, R4, 2   ; Add 2 to y^2
B endIf     ; Branch to endIf
next:
ADD R4, R3, 1   ; R4=f = y + 1 (if x1 <= x2)
endIf:
```



Problem:

We faced many problems during the project because it is a new program that we have not dealt with before, we solved all the problems but we could not check the instruction, we tried many times to implement the instruction but we failed many times, we worked as an integrated team in all stages of the project, we worked face to face and electronically, day and night, and this is what we were able to do, I hope you will consider us and appreciate our efforts on this project, it did not work properly but we learned to work together to solve the problems, and we were able to form an integrated and understanding team. We would like to thank all the doctors, especially Dr. Ibrahim Nimr.