Kuwait University

College of Engineering and Petroleum

Computer Engineering Department

CpE - Course No. 0612368: COMPUTER ORGANIZATION.

Semester: Summer 2022

Section No. 03A

Assignment No. Final Report

Student Name: Yassine Serrid & Abdulrahman Mohammed

Student Id: 2181156439 & 2181147824

Instructor Name: Prof. Sami Habib

TA Name: Eng. Zainab Bahbahani

Date: 11th Of August 2022

Contents

Table Of Figures

Table

1. Problem Description:

In this project we are going to implement and design and ALU (Arithmetic Logic Unit) and a register files system that contains 16 registers each of size 8-bits that can be addressed using 4-bits using Verilog hardware. First, we are going to implement and design an ALU with 4-operations each with its and own binary code that is controlled using the input control that will select the operation to be performed by the ALU, both Input1 and Input2 are 8-bit data bus inputs, the output represent result which is 8-bit data bus from the Input1 and Input2 that will be performed by the ALU. Zero-bit flag is a single bit that will be set to one when the output result that is performed by the ALU equal to zero. In the second part of the project, we are going to design and implement a Read Only Memory (ROM) that stores programs or data that cannot be added to, modified, or deleted. This ROM contains 8-bit address line that represent the number of locations that the ROM has which is 28 = 256 memory location from 00000000 to 11111111 and each address can hold an 8-bits word size. In the second part of this phase, we are going to build the Random Access Memory (RAM) it is the hardware in a computing device where the operating system (OS), application programs and data in current use are kept so they can be quickly reached by the device's processor. It contains 8-bit address line that represent the number of locations that the ROM has which is 28 = 256 memory location from 00000000 to 11111111 and an 8-bits data line to be written to the RAM. In the Third phase we are going to design Architecture RISC-X data path. We are going to implement some basic instructions which are (The R-type includes ADD, AND and OR instructions, The I-type includes adds, lw, sw and beq instruction).With the help of (Quartus II) software we are going to create an ALU Unit which controls the arithmetic and logical operations, Register , ROM which represent the Instruction Memory we can store data in it in the .MIF File,RAM Which represent the Data Memory, A Control Unit which control the signals in the Datapath contains 8 different signals, shift unit, two types of adders and two types of MUXs by using (Verilog HDL) and (Block Diagram Schematic Capture Tool) and testing our project using the wave form.

2. Phase 1 – Designing The First Two Basic Components (ALU – Register File):

2.1. Operations Done By The ALU:

The ALU will perform the following operations:

1- Addition: In this operation input1 will be added to input2 and save the output to result.
2- Subtraction: In this operation input2 will be subtracted from input1 and save the output to result.
3- And: In this operation logic AND will be applied between input1 and input2 and save the output to result.
4- Or: In this operation logic OR will be applied between input1 and input2 and save the output to result.

Table 1. ALU Operations

Operation	Symbol	Binary code
Addition	add	0001
Subtract	sub	0010
Logic And	and	0100
Logic Or	or	1000

In the second part in this phase, we are going to implement and design a register file.

2.2. Register Description:

Register contains these following inputs

1- Read1 : input represent address of register source 1 (rs1) of size 4-bits.
2- Read2: input represent address of register source 2 (rs2) of size.
3- WriteReg : input represent address of register destination (rd) of size 4-bits.
4- Data1: content of register source 1 of size 8-bits.
5- Data2: content of register source 2 of size 8-bits.
6- WriteData: 8bits data to be written on address register specified by WriteReg.
7- RegWrite: control signal that is when equal to 1 the register file will allow 8-bits data at input WriteReg to be written to destination register address specified by WriteReg.

Register 0 with address 0000 must have the value zero all the time and its value cannot be changed.

2.3. Testing cases with expected and actual result for both design

2.3.1 Testing Case 1 : ALU – Expected Output vs Output on Wave Form

Table 2. Expected Output vs Output on Wave Form.

No.	inputs			Expected output		Actual output on waveform	
	Input1	Input2	InputControl	result	Zero signal	result	Zero signal
1	11001010	10111000	0001 add	110000010	0	1 | 10000010	0
2	00111111	00001111	0001 add	01001110	0	01001110	0
3	11001010	10111000	0010 sub	00010010	0	00010010	0
4	00111111	00001111	0010 sub	00110000	0	00110000	0
5	11001010	10111000	0100 and	10001000	0	10001000	0
6	00111111	00001111	0100 and	00001111	0	00001111	0
7	11001010	10111000	1000 or	11111010	0	11111010	0
8	00111111	00001111	1000 or	00111111	0	00111111	0

Input1 + Input2 = Result

1. 11001010 + 10111000 = 110000010 → The output is not their therefore the Zero-flag will not be set to 1.
   Hint : In this addition there is a carry out (overflow) and we can represent the output with just 8-bit, and we will ignore bit number 9.
2. 00111111 + 00001111 = 1001110
3. 11001010 – 10111000 = 0010010
4. 00111111 – 00001111 = 0110000
5. 11001010 & 10111000 = 10001000
6. 00111111 & 10111000 = 10001000
7. 11001010 | 10111000 = 11111010
8. 00111111 | 00001111 = 00111111

Since all the results didn't equal zero then the zero-flag will not be set to 1.

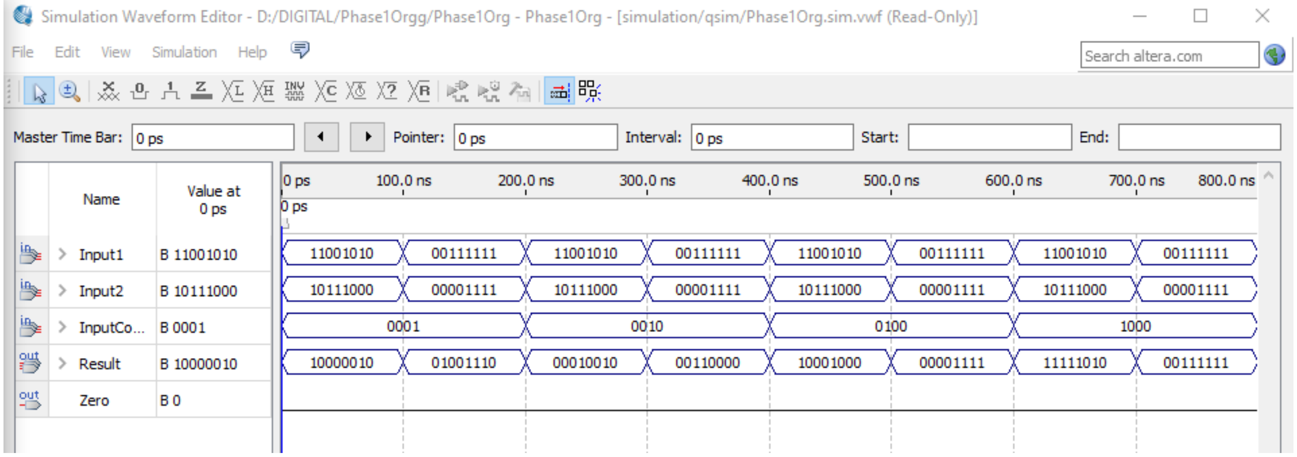
2.3.2. Testing Case 2 : Register Files – Write And Read Procedure:

Table 3. Register Files - Write And Read.

Read1	Read2	Writereg	RegWrite	WriteData	Data1	Data2
		1010	1	00001111		
		1100	1	10111000		
		1000	1	10100011		
		0110	1	10111111		
	1010		0			
1100			0			
1000			0			
	0110		0			

2.4. Screenshot Of Quartus Work:

2.4.1 Testing Case 1 : ALU – Waveform:



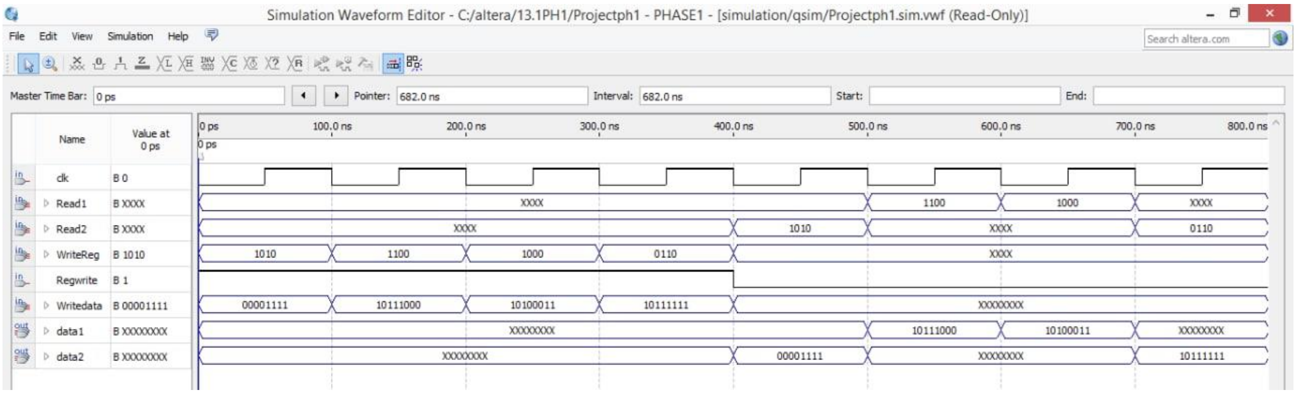
Simulation Waveform Editor - D:/DIGITAL/Phase1Orgg/Phase1Org - Phase1Org - [simulation/qsim/Phase1Org.sim.vwf (Read-Only)]

Master Time Bar: 0 ps Pointer: 0 ps Interval: 0 ps Start: End:

Name	Value at 0 ps	0 ps	100.0 ns	200.0 ns	300.0 ns	400.0 ns	500.0 ns	600.0 ns	700.0 ns
Input1	B 11001010	11001010	00111111	11001010	00111111	11001010	00111111	11001010	00111111
Input2	B 10111000	10111000	00001111	10111000	00001111	10111000	00001111	10111000	00001111
InputCo...	B 0001	0001		0010		0100		1000	
Result	B 10000010	10000010	01001110	00010010	00110000	10001000	00001111	11111010	00111111
Zero	B 0								

Figure 1. Waveform of ALU.

2.4.2. Testing Case 2 : Register Files – Waveform:



Simulation Waveform Editor - C:/altera/13.1PH1/Projectph1 - PHASE1 - [simulation/qsim/Projectph1.sim.vwf (Read-Only)]

Master Time Bar: 0 ps

Pointer: 682.0 ns

Interval: 682.0 ns

Start:

End:

Name	Value at 0 ps
clk	B 0
Read1	B XXXX
Read2	B XXXX
WriteReg	B 1010
Regwrite	B 1
Writedata	B 00001111
data1	B XXXXXXXX
data2	B XXXXXXXX

Figure 2. Waveform of Register File.

2.5. Implementation of the ALU and Register Files Using Verilog:

ALU

```
module ALUnit(Input1,Input2,InputControl,Result,Zero);
    //inputs,outputs and internal variables declared here
    input [7:0] Input1,Input2;
    input [3:0] InputControl;
    output [7:0] Result;
    output reg Zero;
    wire [7:0] Reg1,Reg2;
    reg [7:0] Reg3;

    //Assign A and B to internal variables for doing operations
    assign Reg1 = Input1;
    assign Reg2 = Input2;
    //Assign the output
    assign Result = Reg3;

    //Always block with inputs in the sensitivity list.
    always @(InputControl or Reg1 or Reg2)
    begin
    Reg3 = 0;
        case (InputControl)
         0 : Reg3 = 4'b0000; // Clears The Output
         1 : Reg3 = Reg1 + Reg2; //Addition InputControl:0001
         2 : Reg3 = Reg1 - Reg2;  //Subtraction InpitControl:0010
         4 : Reg3 = Reg1 & Reg2; //LOGIC AND Gate InpitControl:0100
         8 : Reg3 = Reg1 | Reg2; //OR Gate InpitControl:1000
         15 : Reg3 = 4'b1111; // Set The Output To 1
        endcase

    if (Result == 0)
    Zero = 1'b1;
    else
    Zero = 1'b0;
    end

endmodule
```

Register Files

```
module
RegisterFile(clk,Read1,Read2,WriteReg,data1,data2,Writedata,Regwrite);
input clk;  // the write control
input [3:0]Read1,Read2,WriteReg;  // the Register numbers to be read
or written
input[7:0]Writedata;  // data to be written
input Regwrite;  // if zero we can not write data if one we can ,the
write control
output [7:0]data1,data2; // the register values read
reg[7:0] RF[15:0];  // 16 registers each 8 bits long
assign data1= RF[Read1]; //The contents are always available
assign data2= RF[Read2];

always@(posedge clk)begin
if (Regwrite ==1 && WriteReg != 0)
   RF[WriteReg]<= Writedata;
  end

endmodule
```

3. Phase 2 – Designing The Storage Components (RAM – ROM):

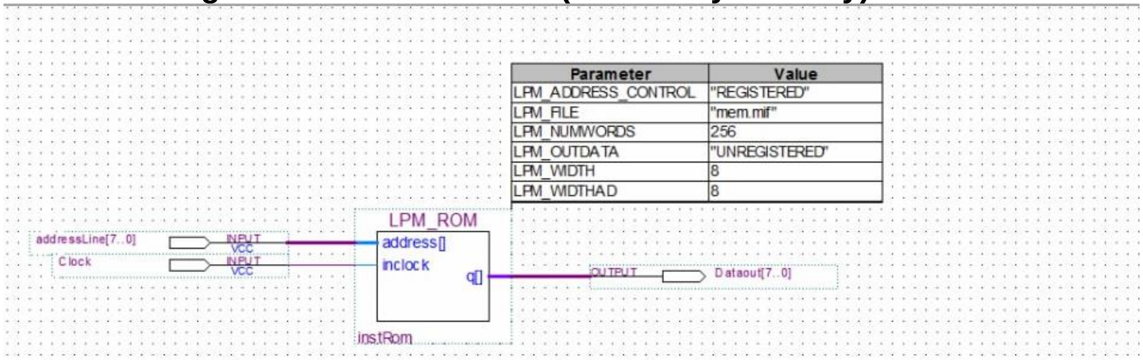
3.1. Read Only Memory (ROM) – Instruction Memory:

In purpose to design a full data path (CPU) we are going to add to the first two components a read only memory that consists of 8-bits address line and each address line can hold data or programs of size 8-bit ;using Quartus block diagram schematic.

- ➢ Requirements to design the ROM are:

1- The RAM has two inputs and one output
2- Input1: 8-bit address line
3- Input2: Clock
4- Memory initialization file (mif) that includes a set of instructions
5- Output1: 8-bit code

3.1.1 Block Diagram Schematic : ROM (Read Only Memory):

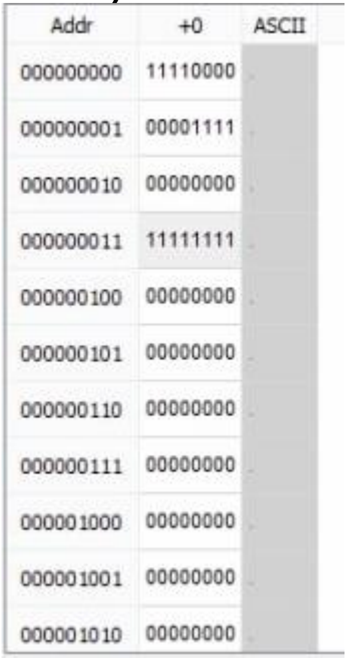


Parameter	Value
LPM_ADDRESS_CONTROL	"REGISTERED"
LPM_FILE	"mem.mif"
LPM_NUMWORDS	256
LPM_OUTDATA	"UNREGISTERED"
LPM_WIDTH	8
LPM_WIDTHAD	8

Figure 3. Block Diagram Schematic of ROM.

As you can see in this screen shot there are two inputs : Address line which is 8-bit input & Clock, In addition an 8-bit output (Dataout).

3.1.2 MIF (Memory Initialization File):



Addr	+0	ASCII
000000000	11110000	
000000001	00001111	
000000010	00000000	
000000011	11111111	
000000100	00000000	
000000101	00000000	
000000110	00000000	
000000111	00000000	
000001000	00000000	
000001001	00000000	
000001010	00000000	

Figure 4. MIF File Of ROM.

3.1.3 Test Case Waveform And Table – ROM:

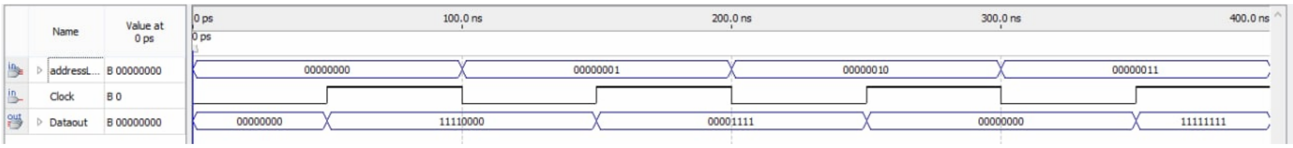


Figure 5. Waveform of ROM.

Table 4. Test Case Table Of ROM.

Address Line	00000000	00000001	00000010	00000011
MIF File Data	11110000	00001111	00000000	00000011
Output	11110000	00001111	00000000	11111111

The same inputs which entered in memory initialization file will appear on the test as output (Dataout) in the positive clock edge.

3.2. Random Access Memory (RAM):

In addition, we are going to add to our small data path a Random Access Memory (RAM).

- ➢ Requirements of RAM:

1- Input1: 8-bits address line.
2- Input2: 8-bits data line to be written to the RAM.
3- Input3: Clock Memory Initialization File (mif) that contains the RAM initialized data (Keep it initialized to zero)
4- Output1: 8-bits data to be read from the RAM that active on positive edge of the clock

- ➢ Control Signals:

1- ReadE: Control signal to enable reading from the RAM.
2- WriteE: Control Signal to enable writing to the RAM.

3.2.1 Block Diagram Schematic : RAM (Random Access Memory):

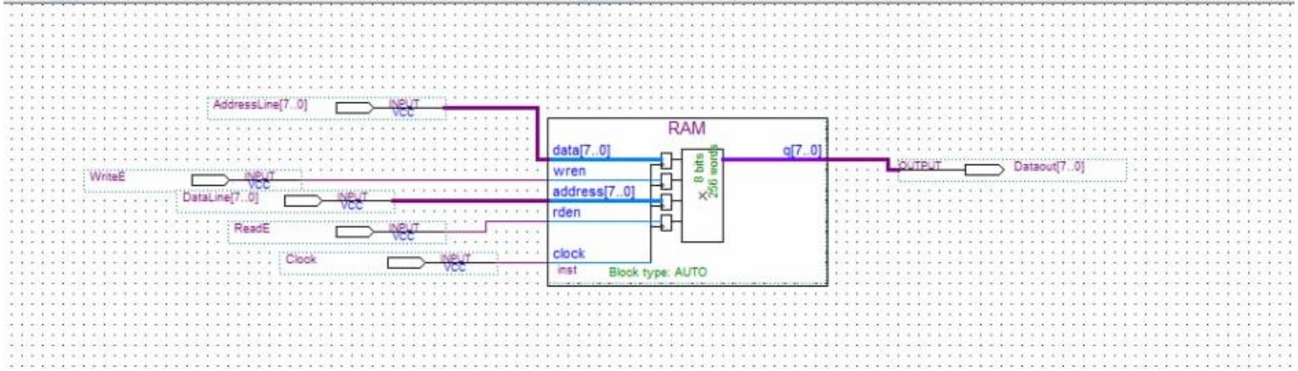


Figure 6. Block Diagram Schematic of RAM.

3.2.2 Test Case Waveform And Table – RAM:

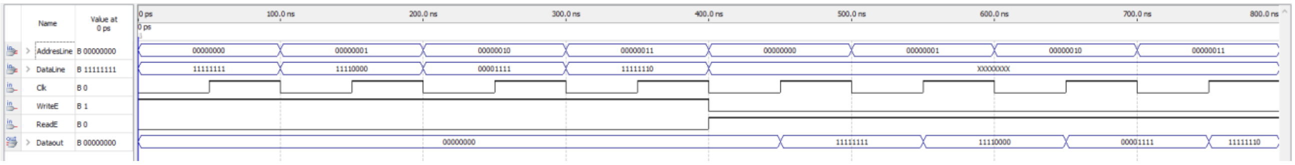


Figure 7. Waveform Of RAM.

Table 5. Test Case Table Of RAM.

Adress Line	00000000	00000001	00000010	00000011	00000000	00000001	00000010	00000011
DataLine	11111111	11110000	00001111	11111110	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
WriteE	1	1	1	1	0	0	0	0
ReadE	0	0	0	0	1	1	1	1
Dataout	00000000				11111111	11110000	00001111	11111110

- ➢ For example: Dataline (11111111) will be stored at Address (00000000) at positive edge since Write Enable signal is set, then the same input will appear on output (Dataout) at the same address when Read Enable signal is set also at positive edge.

4. Phase 3 - Designing The Full Datapath of Architecture RISC-X And Instructions:

4.2. Main functional and control units:

- **Control Unit:** The Control Unit is the unit that enables the signals to be used after reading the opcode.

- **ALU Unit:** The ALU Unit is the unit that handles the arithmetic operations i.e. (addition and subtraction).

- **ALU Control:** The ALU Control is the unit that determines the type of instruction that will be executed i.e. (R-Format & I-Format), and it is specified in detail in phase 1.

- **Register File:** The Register File reads the source registers and the destination registers of the instructions, as well as it supplies the ALU Unit with the values of the sources (data1 and data2) , and it is specified in detail in phase 1.

- **Instruction Memory(ROM) :** In the instruction memory, the Pc supplies it with the address of the instruction to be fetched , and it is specified in detail in phase 2.

- **Data Memory(RAM) :** In the case of the store instruction (sd) data is stored in the data memory, and in the case of load instruction (ld) the data memory outputs the data to the MUX to be written back on the destination register, and it is specified in detail in phase 2.

- **Control Lines :** We have Eight Signal Control Lines (Branch , MemRead , MemtoReg ,ALUOp ,MemWrite , ALUSrc and Regwrite.

4.3. Datapath And Control Signals Design:

Since, can't zoom the full data path is shown in the Quartus II File

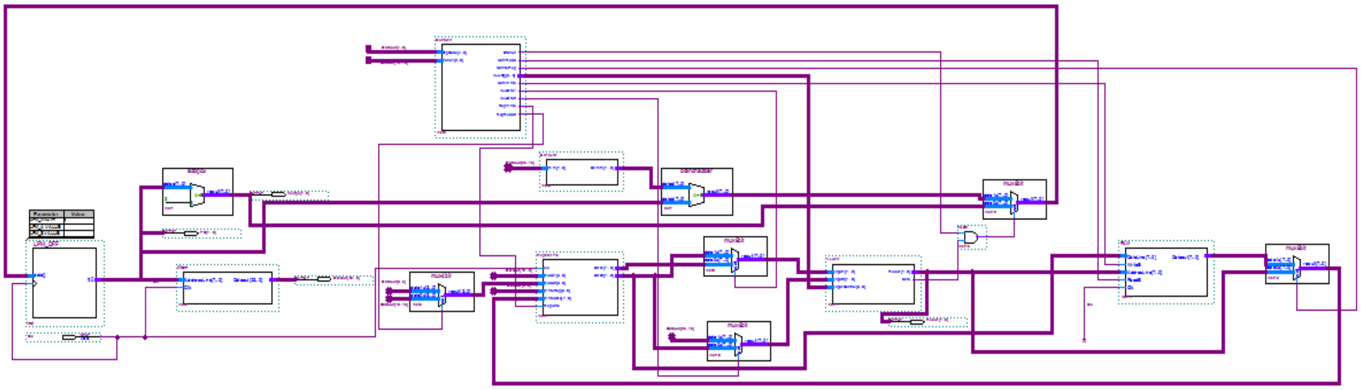


Figure 8. Datapath And Control Signals Design.

4.4. Waveform Of The Datapath:

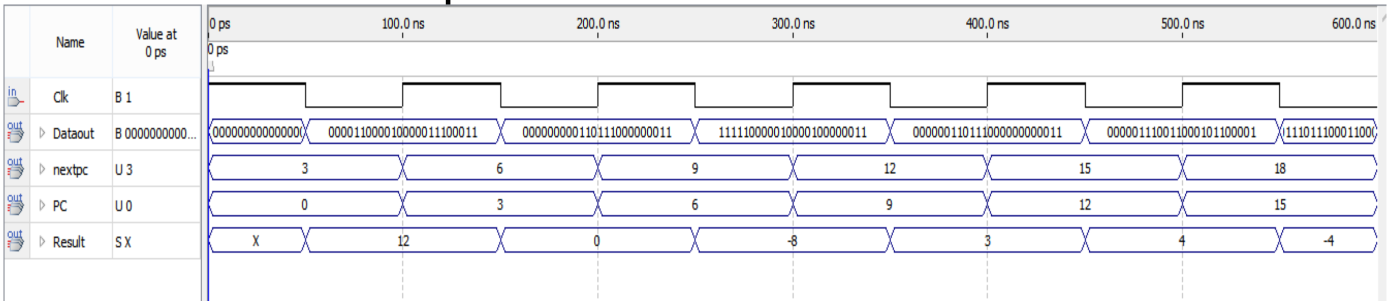


Figure 9. Waveform Of The Datapath.

4.5. Designing The Block Diagram Of The Instruction Memory And Two Adders:

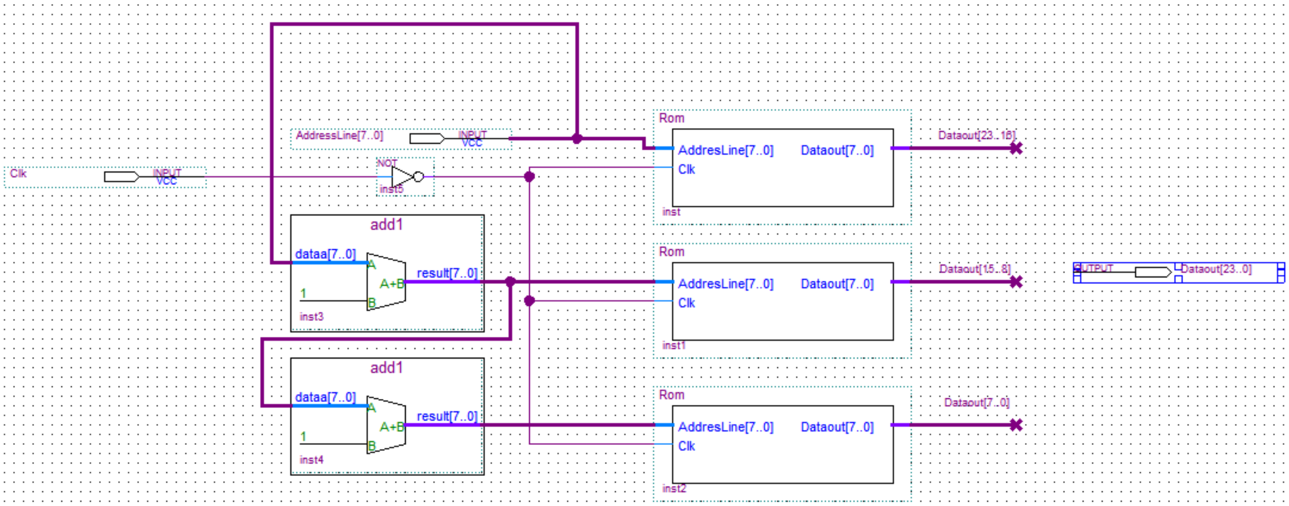
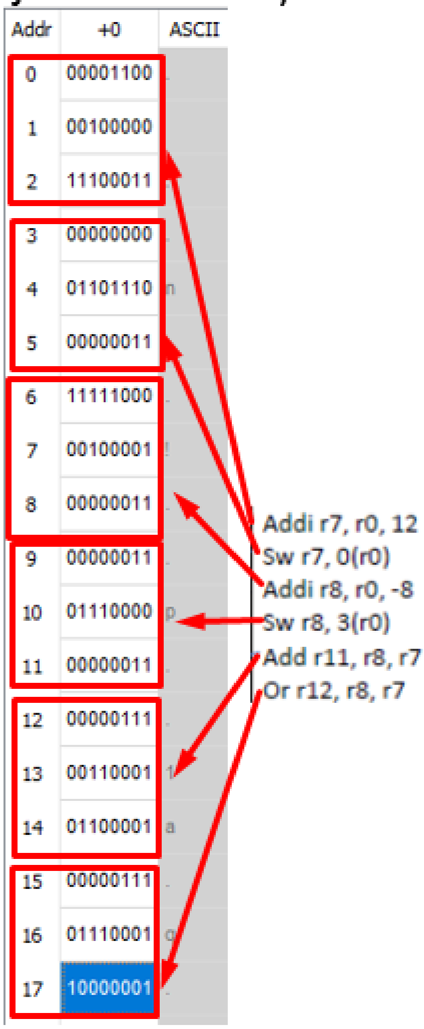


Figure 10. Block Diagram Of The Instruction Memory.

It Is Incremented By 3 Because each Instruction Takes 24 Bit to represent it, And the word size in the instruction memory is 8-bits so we need to increment by 3 to get the full 24-Bits required and get the next instruction.

4.6. MIF (Memory Initialization File) Instruction Memory:



Addr	+0	ASCII
0	00001100	.
1	00100000	
2	11100011	.
3	00000000	.
4	01101110	n
5	00000011	
6	11111000	.
7	00100001	!
8	00000011	.
9	00000011	.
10	01110000	p
11	00000011	.
12	00000111	.
13	00110001	1
14	01100001	a
15	00000111	.
16	01110001	q
17	10000001	.

```
Addi r7, r0, 12
Sw r7, 0(r0)
Addi r8, r0, -8
Sw r8, 3(r0)
Add r11, r8, r7
Or r12, r8, r7
```

Figure 11. Instruction Memory.

Each cluster in the .mif file represent an instruction.

4.7. Verilog Code Needed In Phase 3:

Control Unit

```
module
ContUnit(OpCode,Func1,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUS
rc1,ALUSrc2,RegWrite,RegRead2);
input [4:0]OpCode;
input [2:0]Func1;
output                                                         reg
Branch,MemRead,MemtoReg,MemWrite,ALUSrc1,ALUSrc2,RegWrite,RegRead
2;
output reg [3:0] ALUOp;
always@(OpCode, Func1) begin
    if (OpCode == 5'b00001 && Func1 == 3'b001)begin // ADD
    Branch = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    ALUOp = 4'b0001;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    RegWrite = 1'b1;
    RegRead2  = 1'b0;
    end
    else if (OpCode == 5'b00001 && Func1 == 3'b010)begin // AND
    Branch = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    ALUOp = 4'b0100;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    RegWrite = 1'b1;
    RegRead2  = 1'b0;
    end
    else if (OpCode == 5'b00001 && Func1 == 3'b011)begin // OR
    Branch = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    ALUOp = 4'b1000;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    RegWrite = 1'b1;
    RegRead2  = 1'b0;
    end
    else if (OpCode == 5'b00011 && Func1 == 3'b001)begin // ADDI
    Branch = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'b0;
    ALUOp = 4'b0001;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b1;
    RegWrite = 1'b1;
    RegRead2  = 1'b0;
    end
    else if (OpCode == 5'b00011 && Func1 == 3'b010)begin // LW
    Branch = 1'b0;
    MemRead = 1'b1;
    MemtoReg = 1'b1;
    ALUOp = 4'b0001;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b1;
    RegWrite = 1'b1;
    RegRead2  = 1'b0;
    end
    else if (OpCode == 5'b00011 && Func1 == 3'b011)begin // SW
    Branch = 1'b0;
    MemRead = 1'b0;
    MemtoReg = 1'bx;
    ALUOp = 4'b0001;
    MemWrite = 1'b1;
    ALUSrc1 = 1'b1;
    ALUSrc2 = 1'b1;
    RegWrite = 1'b0;
    RegRead2  = 1'b1;
    end
    else if (OpCode == 5'b00011 && Func1 == 3'b100)begin // BEQ
    Branch = 1'b1;
    MemRead = 1'b0;
    MemtoReg = 1'bx;
    ALUOp = 4'b0010;
    MemWrite = 1'b0;
    ALUSrc1 = 1'b0;
    ALUSrc2 = 1'b0;
    RegWrite = 1'b0;
    RegRead2  = 1'b1;
    end
end
endmodule
```

Shift Unit

```
module ShiftUnit(Imm, ShImm);
input [7:0] Imm;
output reg[7:0] ShImm;
always @ (Imm) begin
ShImm[7:0] = Imm[7:0] * 3;
end
endmodule
```

5. Conclusion:

In this phase of project, we design the first two basic components first one is the ALU that contains 4 different operations each with its specific Input control code consists of 4-bit binary and second one is the register files that contains 16 registers each register is 8-bits long with these two components we are going to build the data path in the next phases using Verilog. In the second phase we designed the next two needed components that we are going to use to build our Datapath, both components are RAM and ROM and testing its functionalities. In the Third phase we will design Architecture RISC-X data path and Control path By merge all of Control Unit, ALU Unit, Register File, Instruction Memory , Data Memory and we need for example some Mux and Adder to Increment the PC this is the Public image for this phase. Through the theoretical and practical tracing of instructions, we have learned how to design and implement a RISC-V processor. I have also learned how to add new instructions to the Datapath.

6. Time Management:

Data & Time	21/July/2022 – 7:30 PM
Meeting Held By	**Yassin Nader – Abdulrahman Mohammed**
1- Understanding Project. 2- Reviewing Verilog Language. 3- Identify Objectives. 4- Implementing The Verilog code of phase 1.	

Data & Time	27/July/2022 – 9:30 PM
Meeting Held By	**Yassin Nader – Abdulrahman Mohammed**
1- Watching The Video posted by Eng. Zainab 2- Reviewing about block Diagram Schematic 3- Building the ROM And ROM and test it using the wave form	

Data & Time	11/August/2022 – 11:30 PM
Meeting Held By	**Yassin Nader – Abdulrahman Mohammed**
1- Watching The Video posted by Eng. Zainab 2- Reviewing about Verilog Language and block Diagram Schematic 3- Design Architecture RISC-X and test it by using waveform	

7. Resources:

[1] ALU, V. (2015). Verilog code for a simple ALU. Retrieved 21 July 2022, from https://verilogcodes.blogspot.com/2015/10/verilog-code-for-simple-alu.html.

[2] S. Brown, Z. Vranesic, “Fundamentals of digital logic with Verilog design,” pp. 220, 2014 Retrieved 21 July 2022.

[3] Lab Manual 264.