CSE112

COMPUTER ORGANIZATION AND ARCHITECTURE

SOPHOMORE CESS

SPRING 2023

MAJOR TASK

Team members	ID
Mohamed Salah Fathy	**21P0117**
Seif Yasser Ahmed	**21P0102**
Youssef Tamer Hossam	**21P0138**
Gaser Zaghlol Hassan	**21P0052**
Ali Tarek Abdelmonim	**21P0123**

Table of Contents

List of Figures:

PHASE 1

Introduction:

This project's requirement is to design a MIPS processor using VHDL that illustrates a basic computer system by simulating the data and control paths, it's done in two phases.

Phase 1 requirements:

1- Implement the MIPS register file that reads simultaneously from two registers and write into another. The implementation should follow the internal logic design of the register file:

- The amin module should be called "RegisterFile".
- The entity should contain the following:
    - **read_sel1** : in std_logic_vector(4 downto 0)
    - **read_sel2** : in std_logic_vector(4 downto 0)
    - **write_sel** : in std_logic_vector(4 downto 0)
    - **write_ena** : in std_logic
    - **clk**: in std_logic
    - **write_data**: in std_logic_vector(31 downto 0)
    - **data1**: out std_logic_vector(31 downto 0)
    - **data2**: out std_logic_vector(31 downto 0)

2- Modify the 32-bit full ALU:

- ALU functional specifications:

ALU Op	Function
0000	AND
0001	OR
0010	ADD
0110	SUB
1100	NOR

- The entity should contain the following:
    - data1: in std_logic_vector (31 downto 0)
    - data2: in std_logic_vector (31 downto 0)
    - aluop: in std_logic_vector (3 downto 0)
    - dataout: out std_logic_vector (31 downto 0)
    - zflag: out std_logic

3- Connect the already-built modules including register file, ALU, to design a simple MIPS CPU Using VHDL. The proposed CPU should be able to perform certain instructions: R-type (AND, OR, ADD, SUB, SLT and NOR).

- The datapath entity should contain the following:
    - Clk, reset : in STD_LOGIC
    - Instr: in STD_LOGIC_VECTOR (31 downto 0)
    - Aluoperation: in STD_LOGIC_VECTOR (3 downto 0)

- Zero: out STD_LOGIC
- Regwrite: in STD_LOGIC
- Aluout: buffer STD_LOGIC_VECTOR (31 downto 0)

Modules' Codes:

ALU module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;
entity ALU is
    Port ( datal : in std_logic_vector(31 downto 0);
data2 : in std_logic_vector(31 downto 0) ;
aluop : in std_logic_vector(3 downto 0) ;
dataout: out std_logic_vector(31 downto 0);
 zflag: out std_logic);
end ALU;
architecture Behavioral of ALU is

Signal result : std_logic_vector(31 downto 0);
begin

process(datal , data2,aluop)
begin
case aluop is
when "0000" => --AND
   result <= datal and data2;
when "0001" => --OR
   result <= datal or data2;
when "0010" => --ADD
   result <= std_logic_vector(unsigned(datal) + unsigned(data2));
when "0110" => --SUB
   result <= std_logic_vector(unsigned(datal) - unsigned(data2));
when "1100" => --NOR
   result <= datal nor data2;
when others => null;
end case;
end process;
dataout<=result;
zflag <='1' when result =x"00000000" else
       '0';
end Behavioral;
```

Figure 1: ALU VHDL code

ALU RTL schematic:

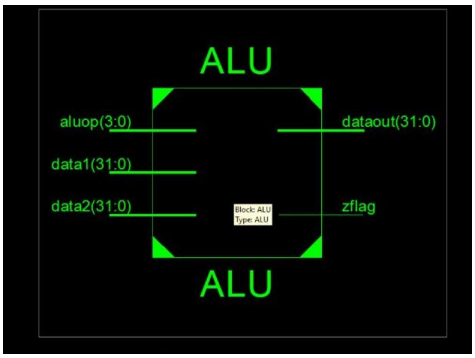

Figure 3: ALU RTL

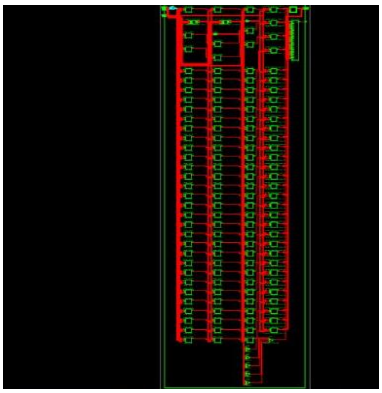

Figure 2: ALU RTL

Multiplexer (MUX) module:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MUX is
    Port ( A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13,A14,A15,A16,A17,A18,A19,A20,A21,A22,A23,A24,A25,A26,A27,A28,A29,A30,A31 : in  STD_LOGIC_VECTOR (31 downto 0);
           Z1 : out  STD_LOGIC_VECTOR (31 downto 0);
           S : in  STD_LOGIC_VECTOR (4 downto 0));
end MUX;

architecture Behavioral of MUX is

begin
 Z1 <=
 A0 when S="00000"  else
 A1 when S="00001" else
 A2 when S="00010" else
 A3 when S="00011" else
 A4 when S="00100" else
 A5 when S="00101" else
 A6 when S="00110" else
 A7 when S="00111" else
 A8 when S="01000" else
 A9 when S="01001" else
 A10 when S="01010" else
 A11 when S="01011" else
 A12 when S="01100" else
 A13 when S="01101" else
 A14 when S="01110" else
 A15 when S="01111" else
 A16 when S="10000" else
 A17 when S="10001" else
 A18 when S="10010" else
 A19 when S="10011" else
 A20 when S="10100" else
 A21 when S="10101" else
 A22 when S="10110" else
 A23 when S="10111" else
```

Figure 4: MUX VHDL code

```
30    A16 when S="10000" else
31    A17 when S="10001" else
32    A18 when S="10010" else
33    A19 when S="10011" else
34    A20 when S="10100" else
35    A21 when S="10101" else
36    A22 when S="10110" else
37    A23 when S="10111" else
38    A24 when S="11000" else
39    A25 when S="11001" else
40    A26 when S="11010" else
41    A27 when S="11011" else
42    A28 when S="11100" else
43    A29 when S="11101" else
44    A30 when S="11110" else
45    A31 when S="11111" ;
46
47  end Behavioral;
48
```

MUX RTL schematic:

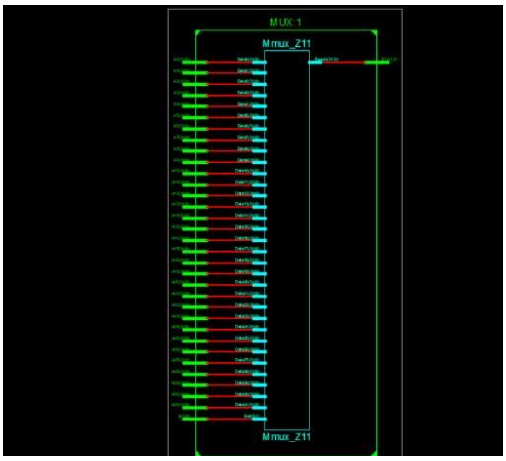

Figure 6 : MUX RTL

Decoder module:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Decodermodule is
    Port ( a : in  STD_LOGIC_VECTOR (4 downto 0);
           y : out  STD_LOGIC_VECTOR (31 downto 0));
end Decodermodule;

architecture Behavioral of Decodermodule is

begin
process(a)
begin
if (a="00000")then
    y <= x"00000001";
elsif (a="00001")then
    y <= x"00000002";
elsif (a="00010")then
    y <= x"00000004";
elsif (a="00011")then
    y <= x"00000008";
elsif (a="00100")then
    y <= x"00000010";
elsif (a="00101")then
    y <= x"00000020";
elsif (a="00110")then
    y <= x"00000040";
elsif (a="00111")then
    y <= x"00000080";
elsif (a="01000")then
    y <= x"00000100";
elsif (a="01001")then
    y <= x"00000200";
elsif (a="01010")then
    y <= x"00000400";
elsif (a="01011")then
    y <= x"00000800";
```

Figure 7 : Decoder VHDL Code

```vhdl
    y <= x"00000800";
elsif (a="01100")then
    y <= x"00001000";
elsif (a="01101")then
    y <= x"00002000";
elsif (a="01110")then
    y <= x"00004000";
elsif (a="01111")then
    y <= x"00008000";
elsif (a="10000")then
    y <= x"00010000";
elsif (a="10001")then
    y <= x"00020000";
elsif (a="10010")then
    y <= x"00040000";
elsif (a="10011")then
    y <= x"00080000";
elsif (a="10100")then
    y <= x"00100000";
elsif (a="10101")then
    y <= x"00200000";
elsif (a="10110")then
    y <= x"00400000";
elsif (a="10111")then
    y <= x"00800000";
elsif (a="11000")then
    y <= x"01000000";
elsif (a="11001")then
    y <= x"02000000";
elsif (a="11010")then
    y <= x"04000000";
elsif (a="11011")then
    y <= x"08000000";
elsif (a="11100")then
    y <= x"10000000";
elsif (a="11101")then
```

Figure 8 : Decoder VHDL Code

Decoder RTL schematic:

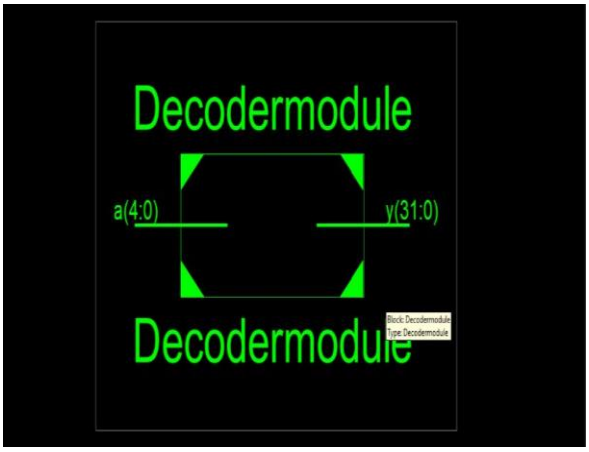

Figure 9 : Decoder RTL

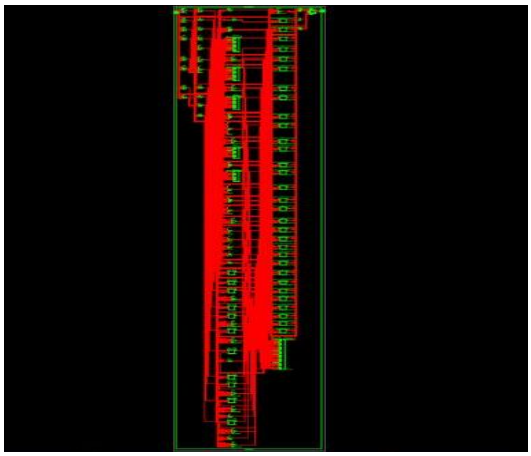

Figure 10 : Decoder RTL

Flop Register module:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity FRegister is
5      Port ( d : in  STD_LOGIC_VECTOR (31 downto 0);
6             rst : in  STD_LOGIC;
7             clk : in  STD_LOGIC;
8             load : in std_logic;
9             q : out  STD_LOGIC_VECTOR (31 downto 0));
10 end FRegister;
11
12 architecture Behavioral of FRegister is
13
14 begin
15 PROCESS(clk,rst,load,d)
16 begin
17 IF(rst='1')THEN
18 q<=(others=>'0');
19 ELSIF(clk'EVENT AND clk='1' and load='1')THEN
20 q<=d;
21 end if;
22 END PROCESS;
23
24 end Behavioral;
```

Figure 11 : Flop Register VHDL Code

Flop register RTL schematic:

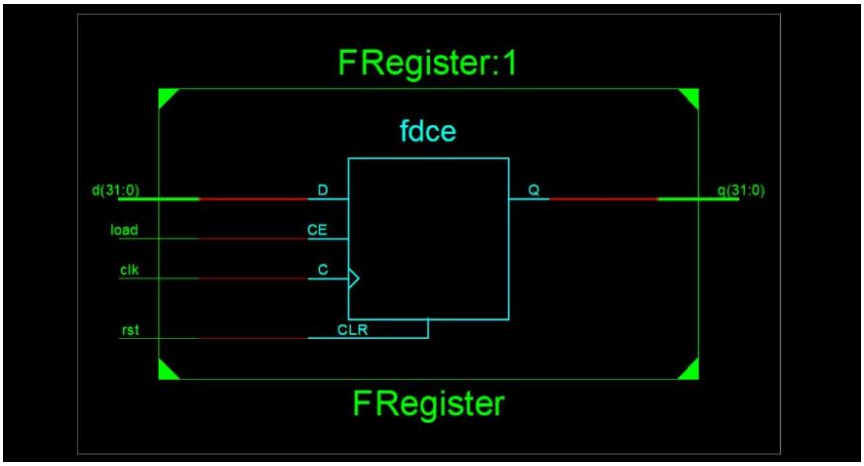

Figure 12 : Flop Register RTL

Register File module:

```
--library UNISIM;
--use UNISIM.VComponents.all;

entity RegisterFile is
port(
read_sel1 : in std_logic_vector(4 downto 0);
read_sel2 : in std_logic_vector(4 downto 0) ;
write_sel : in std_logic_vector(4 downto 0) ;
write_ena : in std_logic ;
clk: in std_logic ;
reset : in std_logic;
write_data: in std_logic_vector(31 downto 0);
data1: out std_logic_vector(31 downto 0) ;
data2: out std_logic_vector(31 downto 0)
);
end RegisterFile;

architecture Behavioral of RegisterFile is

signal out0 : std_logic_vector(31 downto 0);
signal out1 : std_logic_vector(31 downto 0);
signal out2 : std_logic_vector(31 downto 0);
signal out3 : std_logic_vector(31 downto 0);
signal out4 : std_logic_vector(31 downto 0);
signal out5 : std_logic_vector(31 downto 0);
signal out6 : std_logic_vector(31 downto 0);
signal out7 : std_logic_vector(31 downto 0);
signal out8 : std_logic_vector(31 downto 0);
signal out9 : std_logic_vector(31 downto 0);
signal out10 : std_logic_vector(31 downto 0);
signal out11 : std_logic_vector(31 downto 0);
signal out12 : std_logic_vector(31 downto 0);
signal out13 : std_logic_vector(31 downto 0);
signal out14 : std_logic_vector(31 downto 0);
```

Figure 14:reg file code1

```
signal load27 : std_logic;
signal load28 : std_logic;
signal load29 : std_logic;
signal load30 : std_logic;
signal load31 : std_logic;
signal dec_out: std_logic_vector(31 downto 0);


begin


load0 <= write_ena and dec_out(0);
load1 <= write_ena and dec_out(1);
load2 <= write_ena and dec_out(2);
load3 <= write_ena and dec_out(3);
load4 <= write_ena and dec_out(4);
load5 <= write_ena and dec_out(5);
load6 <= write_ena and dec_out(6);
load7 <= write_ena and dec_out(7);
load8 <= write_ena and dec_out(8);
load9 <= write_ena and dec_out(9);
load10 <= write_ena and dec_out(10);
load11 <= write_ena and dec_out(11);
load12 <= write_ena and dec_out(12);
load13 <= write_ena and dec_out(13);
load14 <= write_ena and dec_out(14);
load15 <= write_ena and dec_out(15);
load16 <= write_ena and dec_out(16);
load17 <= write_ena and dec_out(17);
load18 <= write_ena and dec_out(18);
load19 <= write_ena and dec_out(19);
load20 <= write_ena and dec_out(20);
load21 <= write_ena and dec_out(21);
load22 <= write_ena and dec_out(22);
load23 <= write_ena and dec_out(23);
load24 <= write_ena and dec_out(24);
```

Figure 13:reg file code2

Register file RTL schematic:

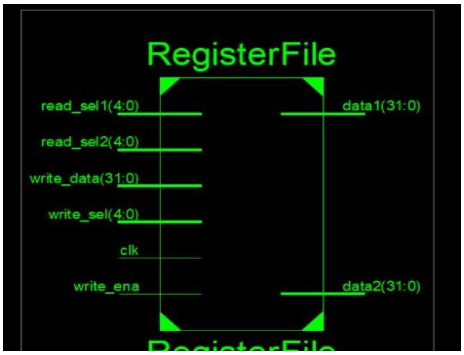

Figure 16 : Register File RTL

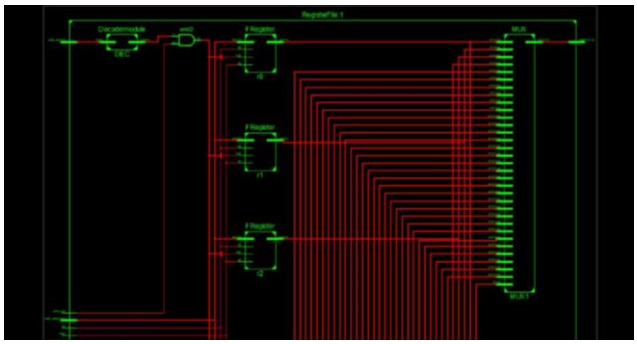

Figure 15: Register File RTL

R Format Module:

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.mainmodulepack.all;
5
6  entity Phase1module is
7  port(
8  clk, reset: in STD_LOGIC;
9  instr: in STD_LOGIC_VECTOR(31 downto 0);
10 aluoperation: in STD_LOGIC_VECTOR(3 downto 0);
11 zero: out STD_LOGIC;
12 regwrite: in STD_LOGIC;
13 aluout : buffer STD_LOGIC_VECTOR(31 downto 0)
14 );
15 end Phase1module;
16
17 architecture Behavioral of Phase1module is
18
19 signal datalout : STD_LOGIC_VECTOR(31 downto 0);
20 signal data2out : STD_LOGIC_VECTOR(31 downto 0);
21
22
23 begin
24
25 RegFile : RegisterFile port map(instr(25 downto 21),instr(20 downto 16),instr(15 downto 11),regwrite,clk,aluout,datalout,data2out);
26 ALU1 : ALU port map(datalout,data2out,aluoperation,aluout,zero);
27 end Behavioral;
28
```

Figure 17 : R-Format VHDL Code

R Format RTL Schematic:

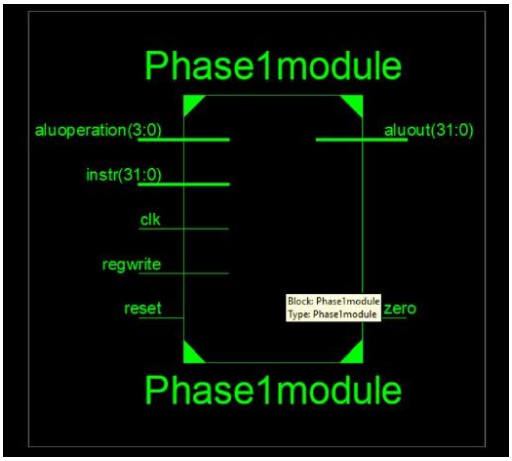

Figure 19 : R-Format RTL

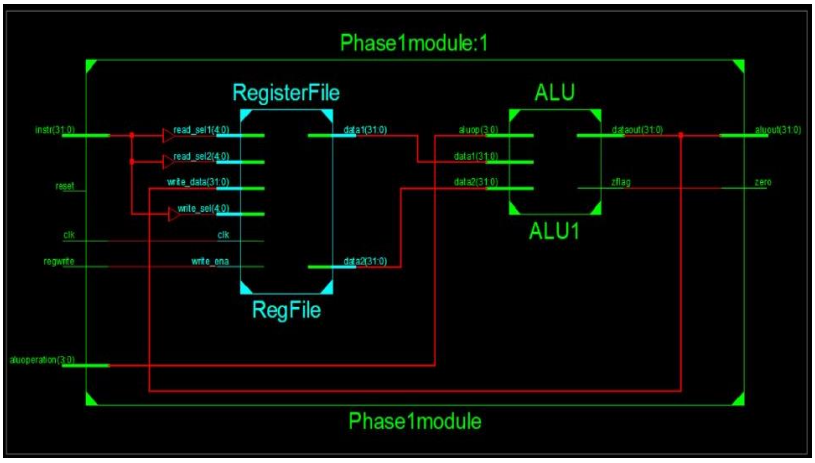

Figure 18 : R-Format RTL

Code Description:

- In Phase 1 of MIPS Processor Implementation using VHDL we implemented the R-Format Instructions only.
- We used Decoder, Multiplexers, and Flop Registers to implement the Register File
- We also implemented an ALU that has AND, OR ADD, SUB, and, NOR operations.
- The Register File has two modes which are read and write.
- In the write mode we used 32 Flop Register and a decoder.
- In the read mode we used 32 Flop Registers and 2 Multiplexers.
- We used all of these components by declaring them as a package named as mainmodulepack then, declaring them in other modules using use work.mainmodulepack.all
- The load function were implemented by implementing 32 Signals

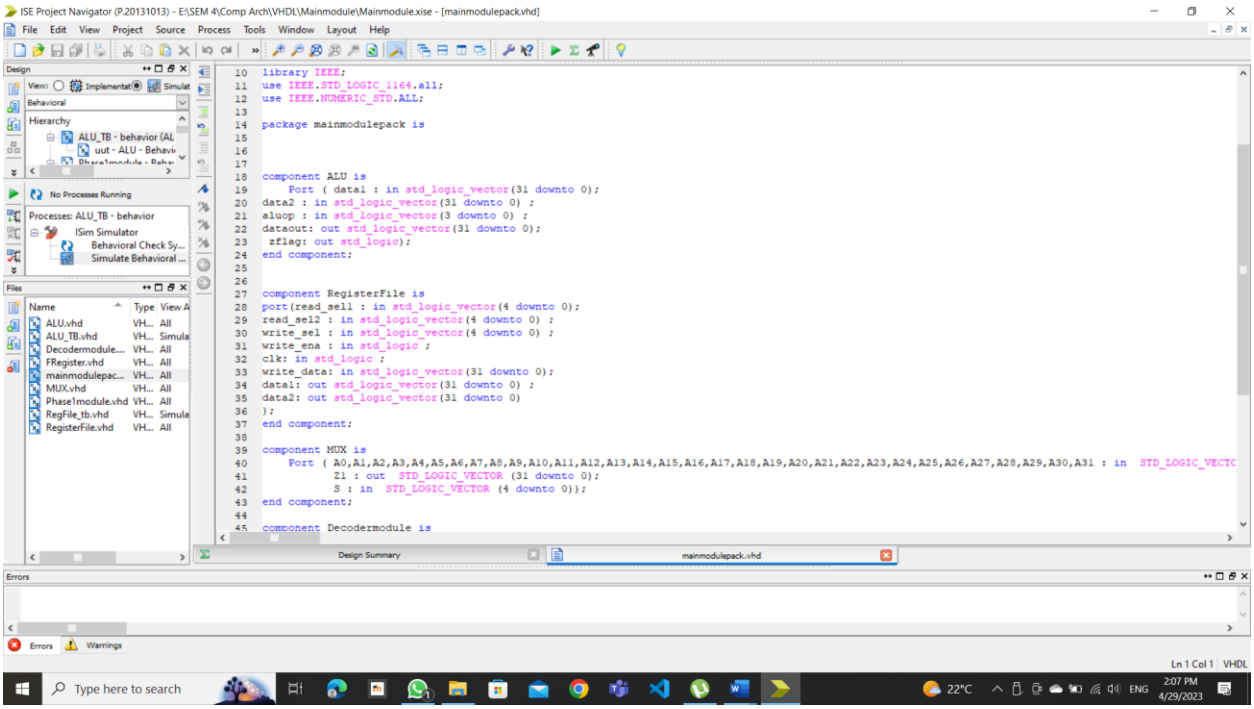

Figure 20: Main Module Pack (mainmodulepack)

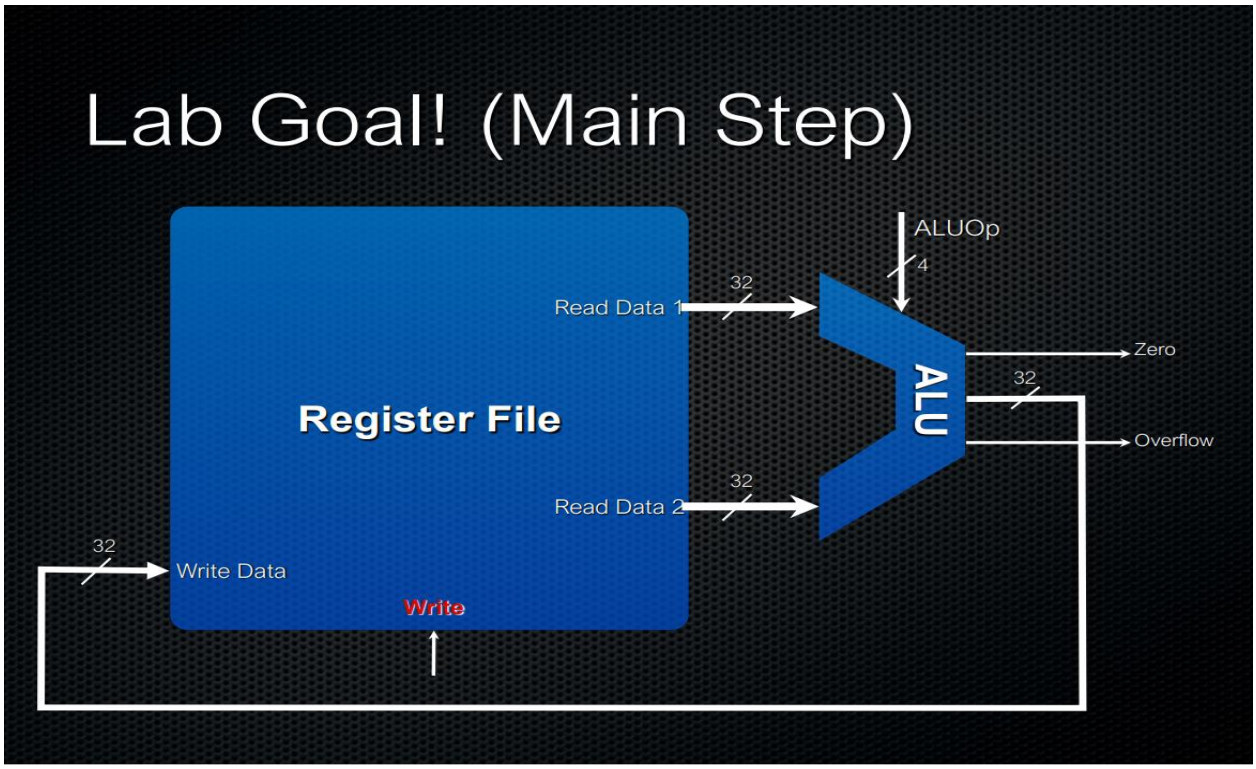

Figure 21

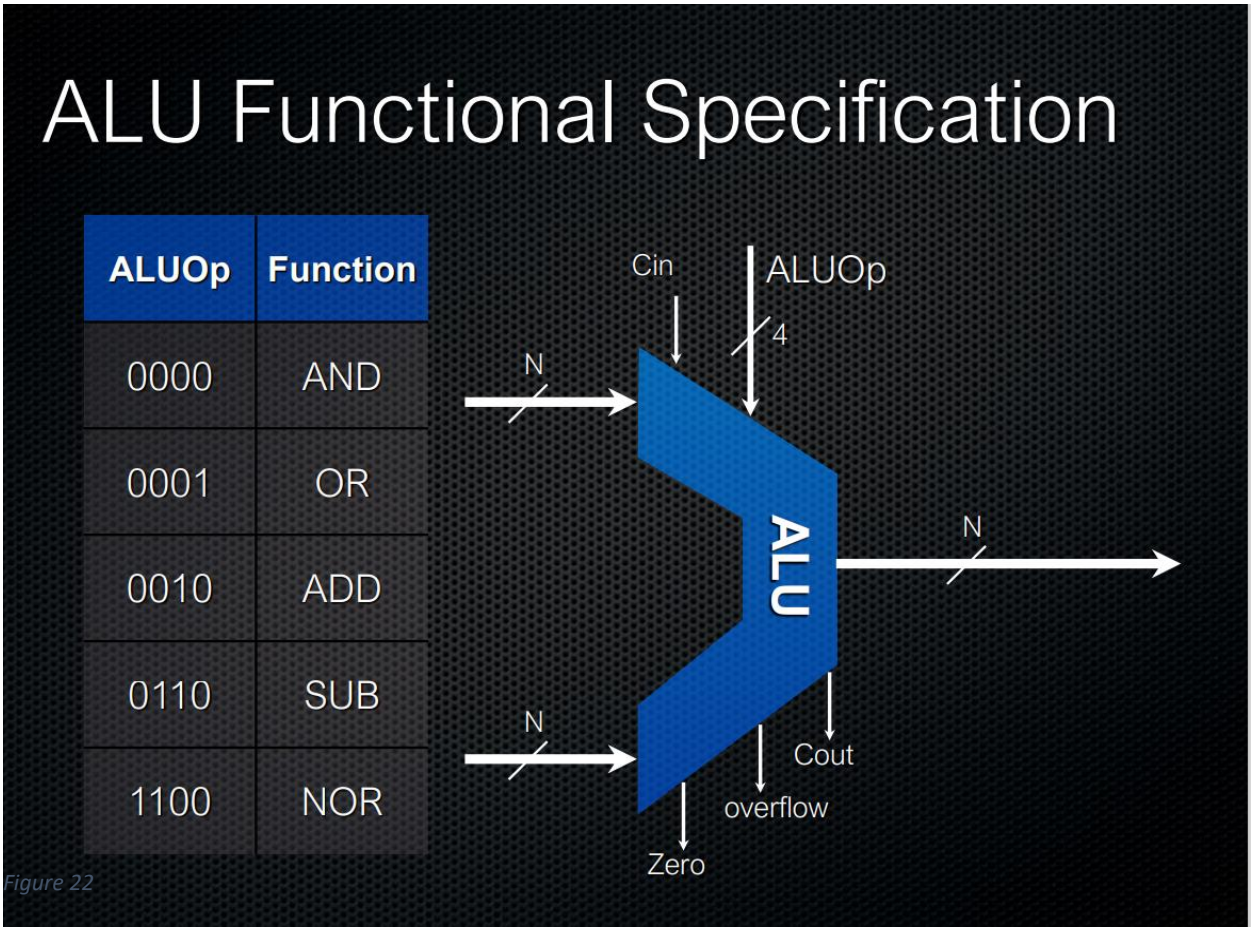

Figure 22

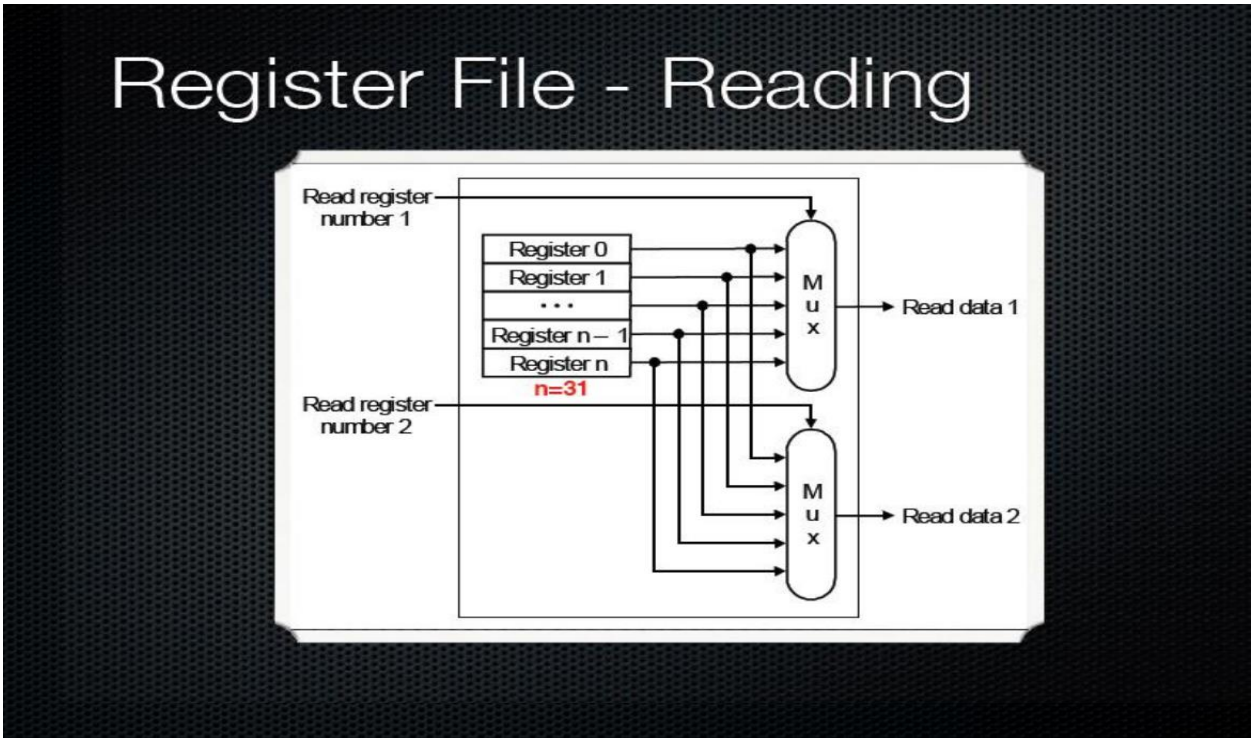

Figure 24

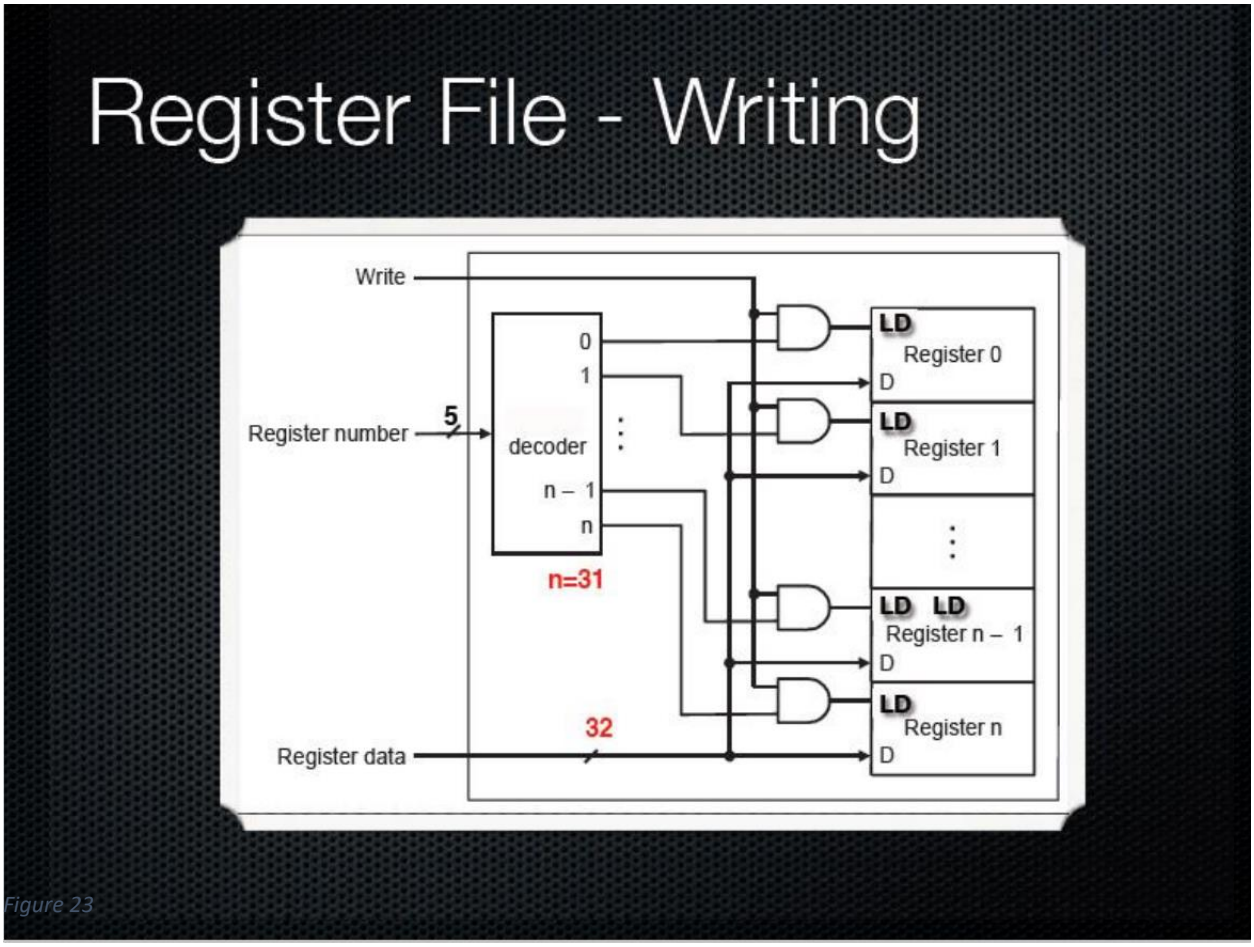

Figure 23

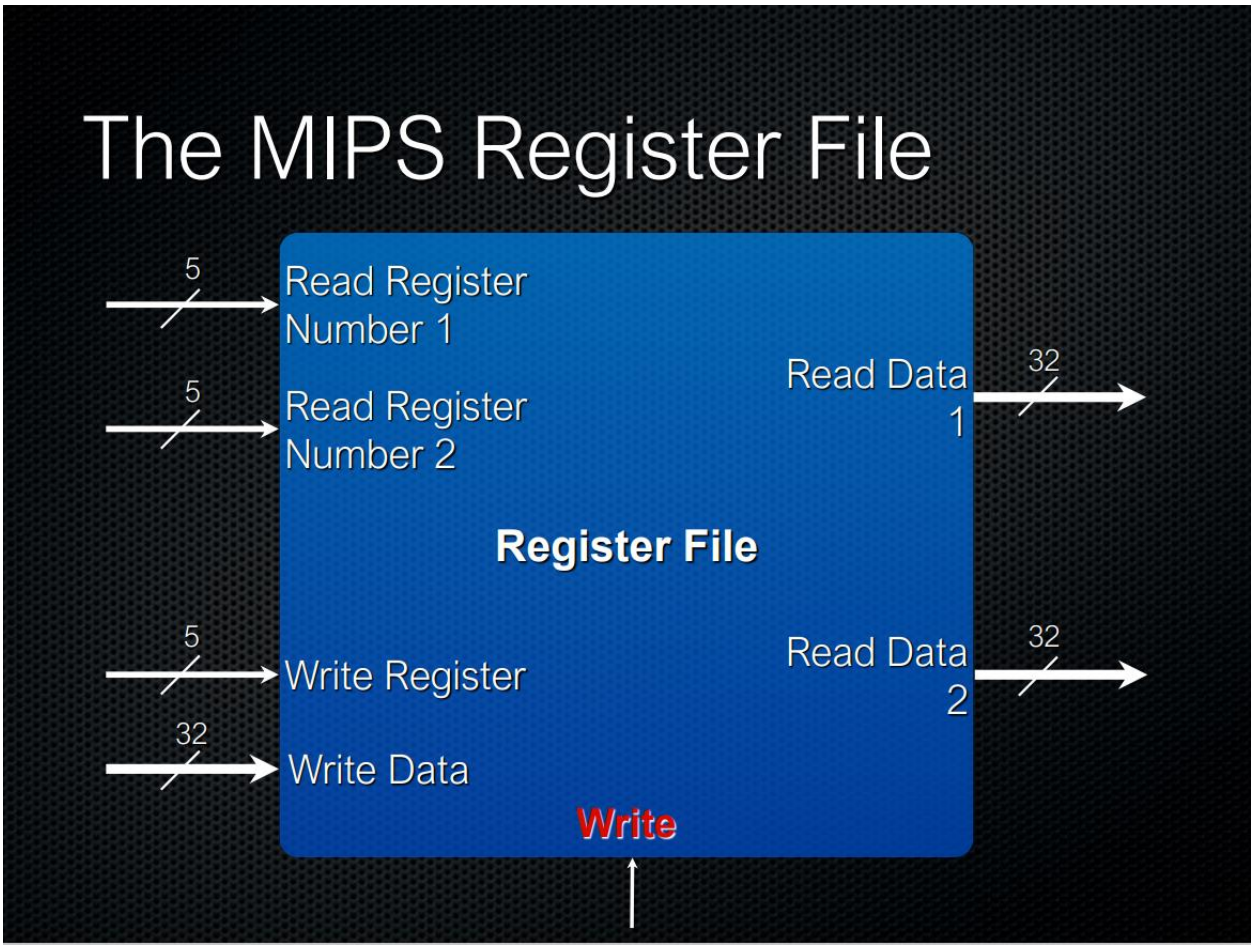
The MIPS Register File
5
Read Register Number 1
5
Read Register Number 2
Register File
Read Data 1
32
5
Write Register
32
Write Data
Read Data 2
32
Write

Figure 25

PHASE 2

Phase 2 Requirements:

To Modify the MIPS CPU as to be able to perform not only R instructions, but also I-type (lw, sw, beq) and J instruction.

Steps:

1. Implement the control module, which is responsible for all the control signals.
2. Implement the Mips module by connecting the datapath with the control module.
3. Connect the Mips module with instruction and data memory module together.
4. Then Fill the memory module by a simple program.
5. The CPU should be able to execute this program.
6. Simulate the results and check the final results.

This diagram shows the abstract CPU design.

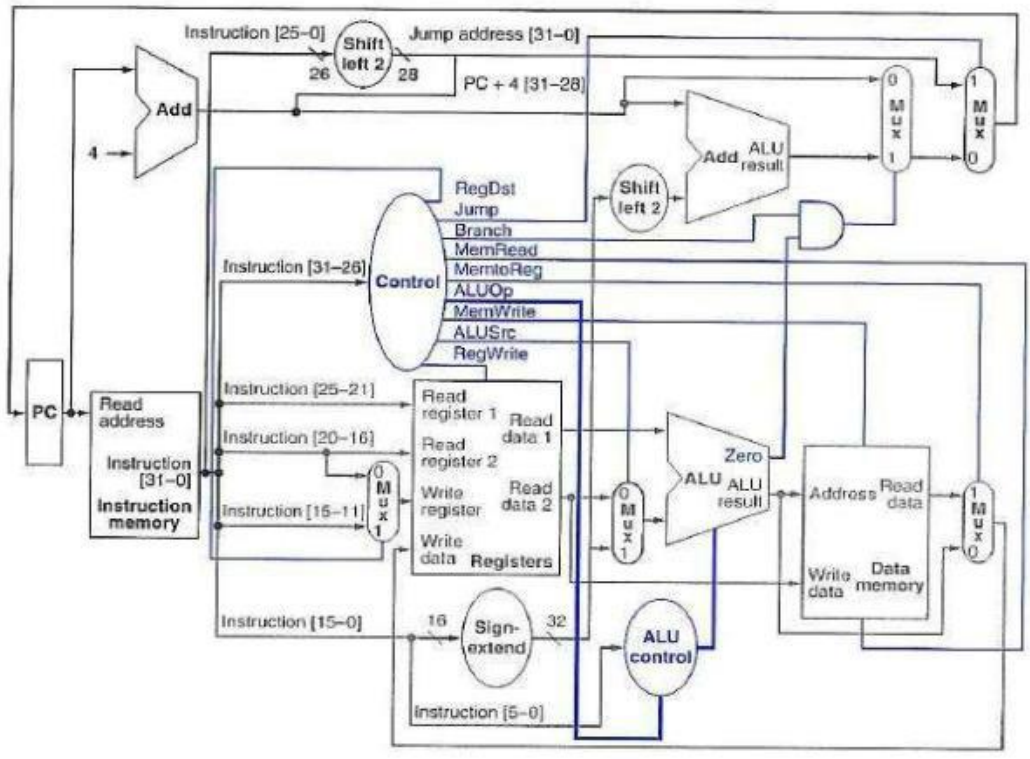

Figure 26:cpu design

7. The main module entity contains the following:
    - CLK: IN STD_LOGIC;
    - RST: IN STD_LOGIC;
    - Writedata,dataadr: OUT STD_LOGIC_VECTOR(31 downto 0);
    - memwrite: OUT STD_LOGIC;
8. Test Case:

```
main:   addi $2, $0, 5 # initialize $2 = 5 0 20020005
        addi $3, $0, 12 # initialize $3 = 12 4 2003000c
        addi $7, $3, -9 # initialize $7 = 3 8 2067fff7
        or $4, $7, $2 # $4 = (3 OR 5) = 7 c 00e22025
        and $5, $3, $4 # $5 = (12 AND 7) = 4 10 00642824
        add $5, $5, $4 # $5 = 4 + 7 = 11 14 00a42820
        beq $5, $7, end # shouldn't be taken 18 10a7000a
        slt $4, $3, $4 # $4 = 12 < 7 = 0 1c 0064202a
        beq $4, $0, around # should be taken 20 10800001
        addi $5, $0, 0 # shouldn't happen 24 20050000
around: slt $4, $7, $2 # $4 = 3 < 5 = 1 28 00e2202a
        add $7, $4, $5 # $7 = 1 + 11 = 12 2c 00853820
        sub $7, $7, $2 # $7 = 12 - 5 = 7 30 00e23822
        sw $7, 68($3) # [80] = 7 34 ac670044
        lw $2, 80($0) # $2 = [80] = 7 38 8c020050
        j end # should be taken 3c 08000011
        addi $2, $0, 1 # shouldn't happen 40 20020001
end:    sw $2, 84($0) # write mem[84] = 7 44 ac020054
```

```
Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84
```

Figure 27:test case

Modules' Codes:

Data path:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.mainmodulepack.all;

entity DataPath is
port ( clk : in std_logic;
       reset : in std_logic;
       readdata : in std_logic_vector(31 downto 0);
       instr : in std_logic_vector(31 downto 0);
       memtoreg , pcsrc , alusrc , regwrite , regdst : in std_logic;
       aluoperation : in std_logic_vector(3 downto 0);
       zero : out std_logic;
       pc : out std_logic_vector(31 downto 0);
       jump : in std_logic;
       aluout , writedata : out std_logic_vector(31 downto 0));
end DataPath;

architecture Behavioral of DataPath is

signal writereg : std_logic_vector(4 downto 0);
signal pcjump , pcnext , pcnextbr , pcplus4 , pcbranch : std_logic_vector(31 downto 0);
signal signimm , signimmsh : std_logic_vector(31 downto 0);
signal load : std_logic;
signal datalout : STD_LOGIC_VECTOR(31 downto 0);
signal data2out : STD_LOGIC_VECTOR(31 downto 0);
signal writedata1 : std_logic_vector(31 downto 0);
signal pc1 : std_logic_vector(31 downto 0);
signal aluout1 : std_logic_vector(31 downto 0);
signal writedata2 : std_logic_vector(31 downto 0);
begin

load <= '1' ;

pc <= pc1 ;

aluout <= aluout1 ;

writedata <= writedata2 ;
```

Figure 28:data path code1

```
aluout <= aluout1 ;

writedata <= writedata2 ;

pcjump <= pcplus4(31 downto 28) & instr(25 downto 0) & "00";

pcreg : FRegister port map( pcnext , reset , clk , load , pc1 );

pcadd1: adder port map( pc1 , X"00000004" , pcplus4);

immsh : Shifter port map( signimm , signimmsh);

pcadd2 : adder port map( pcplus4 , signimmsh , pcbranch );

pcbrmux : MUX2x1 port map(pcplus4 , pcbranch , pcnextbr , pcsrc );

pcmux : MUX2x1 port map(pcnextbr , pcjump ,  pcnext , jump );

signext : SignedBits port map(instr(15 downto 0) , signimm );

regfilemux : MUX2x15bits port map(instr(20 downto 16)  , instr(15 downto 11)  , writereg , regdst);
RegFile : RegisterFile port map(instr(25 downto 21),instr(20 downto 16),writereg,regwrite,clk,writedata1,datalout, writedata2 );

alumux : MUX2x1 port map (writedata2 , signimm , data2out , alusrc );
ALU1 : ALU port map(datalout , data2out , aluoperation , aluout1 ,zero);

memregmux : MUX2x1 port map(aluout1 , readdata  , writedata1 , memtoreg);



end Behavioral;
```

Figure 29:data path code2

Data path RTL schematic:

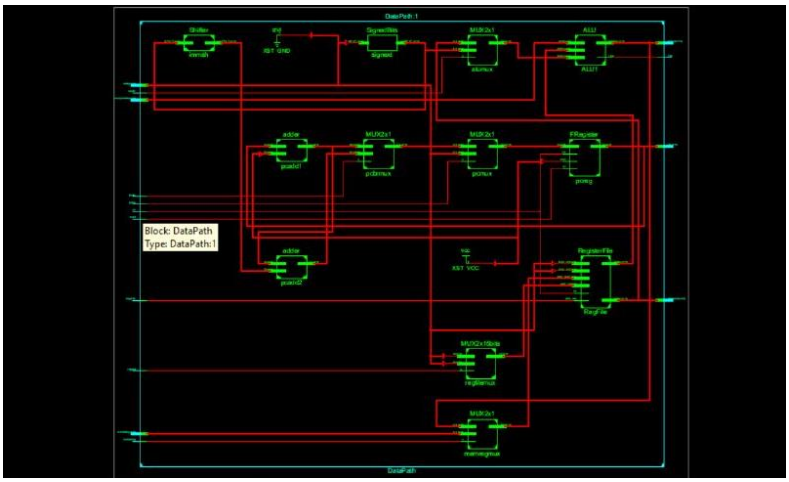

Figure 30:data path RTL schematic

ALU decoder:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity AluDecoder is
5  port(funct : in std_logic_vector(5 downto 0);
6       aluop : in std_logic_vector(1 downto 0);
7       alucontrol : out std_logic_vector(3 downto 0));
8  end AluDecoder;
9
10 architecture Behavioral of AluDecoder is
11
12 begin
13 process(aluop , funct)
14 begin
15 case aluop is
16    when "00" => alucontrol <="0010";
17    when "01" => alucontrol <="0110";
18    when others =>
19    case funct is
20    when "100000" => alucontrol <= "0010" ;
21    when "100010" => alucontrol <= "0110" ;
22    when "100100" => alucontrol <= "0000" ;
23    when "100101" => alucontrol <= "0001" ;
24    when "101010" => alucontrol <= "0111" ;
25    when others => alucontrol <= "----";
26    end case ;
27    end case ;
28    end process;
29
30 end Behavioral;
31
32
```

Figure 31:ALU decoder code

ALU decoder RTL schematic:

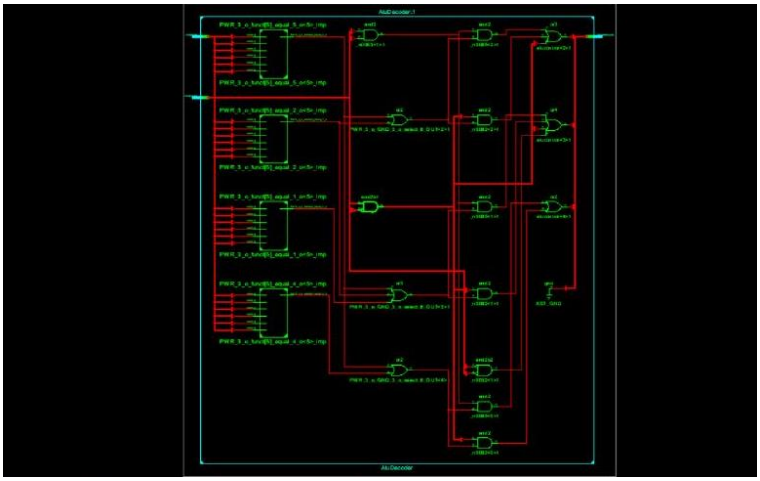

Figure 32:ALU decoder RTL schematic

Main decoder:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity MainDecoder is
5  port ( op : in std_logic_vector(5 downto 0);
6         memtoreg , memwrite , branch , alusrc , regdst , regwrite , jump : out std_logic ;
7         aluop : out std_logic_vector(1 downto 0));
8  end MainDecoder;
9
10 architecture Behavioral of MainDecoder is
11 signal controls : std_logic_vector(8 downto 0);
12 begin
13
14 process(op)
15 begin
16 case op is
17    when"000000" => controls <= "110000010" ;
18    when"100011" => controls <= "101001000" ;
19    when"101011" => controls <= "001010000" ;
20    when"000100" => controls <= "000100001" ;
21    when"001000" => controls <= "101000000" ;
22    when"000010" => controls <= "000000100" ;
23    when others => controls <= "---------";
24 end case ;
25 end process;
26 (regwrite , regdst , alusrc , branch , memwrite , memtoreg , jump , aluop(1),aluop(0))<= controls;
27 end Behavioral;
28
29
```

Figure 33:main decoder code

Main decoder RTL schematic:

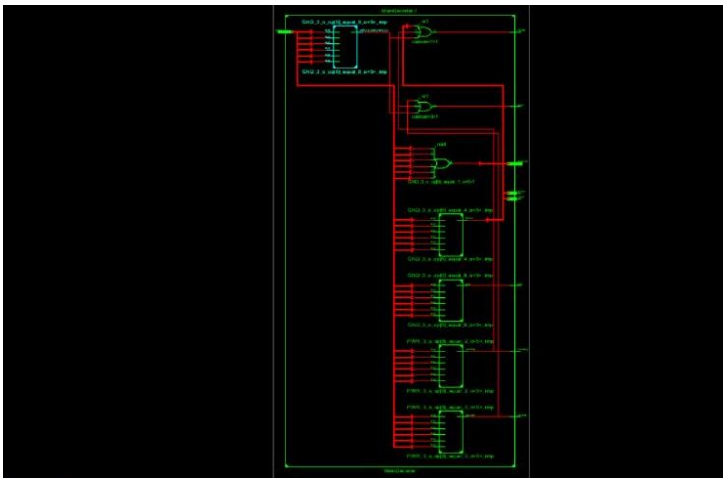

Figure 34:main decoder RTL schematic

Controller unit:

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.ControllerPackage.all;
5
6  entity ControllerUnit is
7  port( op , funct : in std_logic_vector(5 downto 0);
8        zero : in std_logic ;
9        memtoreg , memwrite , pcsrc , alusrc , regdst , regwrite , jump : out std_logic;
10       alucontrol : out std_logic_vector(3 downto 0));
11 end ControllerUnit;
12
13 architecture Behavioral of ControllerUnit is
14 signal aluop : std_logic_vector(1 downto 0);
15 signal branch : std_logic ;
16 begin
17
18 maindec1 : MainDecoder port map(op , memtoreg , memwrite , branch , alusrc , regdst , regwrite , jump , aluop);
19 Aludec1 : AluDecoder port map(funct , aluop , alucontrol);
20 pcsrc <= branch and zero ;
21
22 end Behavioral;
23
24
```

Figure 35:controller unit code

Controller unit RTL schematic:

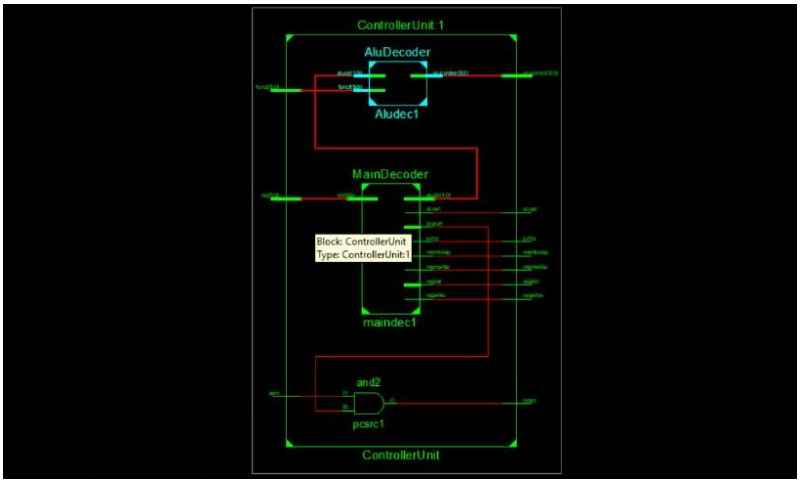

Figure 36:controller unit RTL schematic

MIPS:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.mainmodulepack.all;

entity MIPS is
port( clk , reset : in std_logic;
       pc : out std_logic_vector(31 downto 0);
       instr : in std_logic_vector(31 downto 0);
       memwrite : out std_logic;
       aluout , writedata : out std_logic_vector(31 downto 0);
       readdata : in std_logic_vector(31 downto 0));
end MIPS;

architecture Behavioral of MIPS is

signal memtoreg , alusrc ,regdst , regwrite , jump , pcsrc: std_logic;
signal zero : std_logic;
signal alucontrol : std_logic_vector(3 downto 0);

begin

Controller : ControllerUnit port map(instr(31 downto 26) , instr(5 downto 0), zero , memtoreg , memwrite , pcsrc , alusrc , regdst , regwrite , jump , alucontrol);

datapath1 : DataPath port map ( clk , reset , readdata , instr , memtoreg , pcsrc , alusrc , regwrite , regdst , alucontrol , zero , pc , jump , aluout , writedata )


end Behavioral;

```

Figure 37:mips code

MIPS RTL schematic:

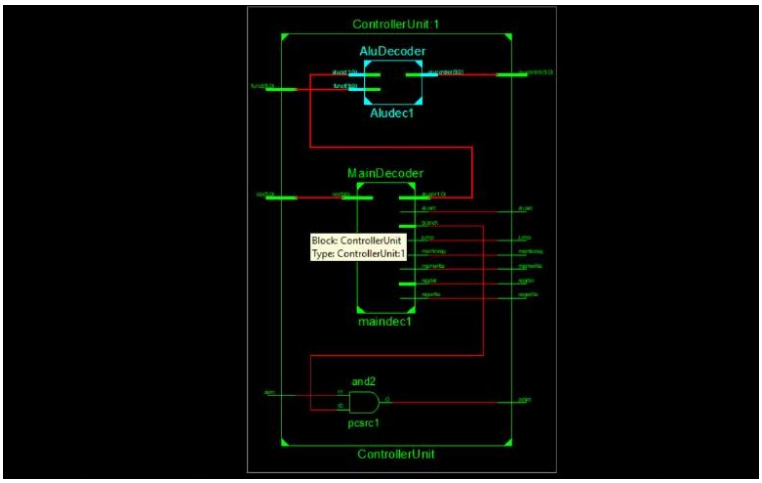

Figure 38:mips RTL schematic

Main module:

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use work.mainmodulepack.all;
5
6  entity MainModule is
7  port( clk , reset : in std_logic;
8        writedata , dataadr : out std_logic_vector(31 downto 0);
9        memwrite : out std_logic);
10 end MainModule;
11
12 architecture Behavioral of MainModule is
13
14 signal memwritet : std_logic;
15 signal pc , instr , readdata , dataadrt , writedatat : std_logic_vector(31 downto 0);
16 signal aluout : std_logic_vector(31 downto 0 );
17
18 begin
19
20
21 memwrite <= memwritet ;
22
23 dataadr <= aluout;
24
25 writedata <= writedatat ;
26
27 mips1 : MIPS port map(clk , reset , pc , instr , memwritet , aluout , writedatat , readdata );
28
29 instrmem : imem port map(pc(7 downto 2) , instr);
30
31 datamem : dmem port map(clk , memwritet , aluout , writedatat , readdata );
32
33
34 end Behavioral;
35
```

Figure 39:main module code

main module RTL schematic:

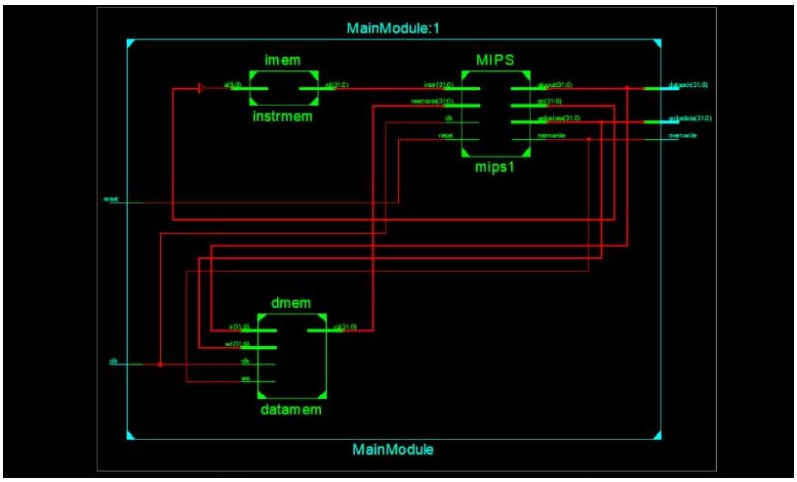

Figure 40:main module RTL schematic

Main module_tb(test bench):

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;
use ieee.numeric_std.all;

ENTITY Mainmdoule_tb IS
END Mainmdoule_tb;

ARCHITECTURE behavior OF Mainmdoule_tb IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT MainModule
port(clk, reset: in STD_LOGIC;
writedata, dataadr: out STD_LOGIC_VECTOR(31 downto 0);
memwrite: out STD_LOGIC);
end component;


signal writedata, dataadr: STD_LOGIC_VECTOR(31 downto 0);
signal clk, reset, memwrite: STD_LOGIC;
   -- Clock period definitions
   constant clk_period : time := 10 ns;

BEGIN
   dut: MainModule port map(clk, reset, writedata, dataadr, memwrite);
-- Generate clock with 10 ns period
process begin
clk <= '1';
wait for 5 ns;
clk <= '0';
wait for 5 ns;
end process;
-- Generate reset for first two clock cycles
process begin
reset <= '1';
wait for 22 ns;
reset <= '0';
wait;
```

Figure 41:main module test bench code1

```
wait;
end process;

-- check that 7 gets written to address 84 at end of program
process(clk) begin
if (clk'event and clk = '0' and memwrite = '1') then
if (CONV_INTEGER(dataadr) = 84 and CONV_INTEGER(writedata) = 7) then
report "NO ERRORS: Simulation succeeded" severity failure;
elsif (CONV_INTEGER(dataadr) = 84) then
report "Simulation failed" severity failure;
end if;
end if;

end process;
end;

```

Figure 42:main module test bench code2

Main module test bench results:

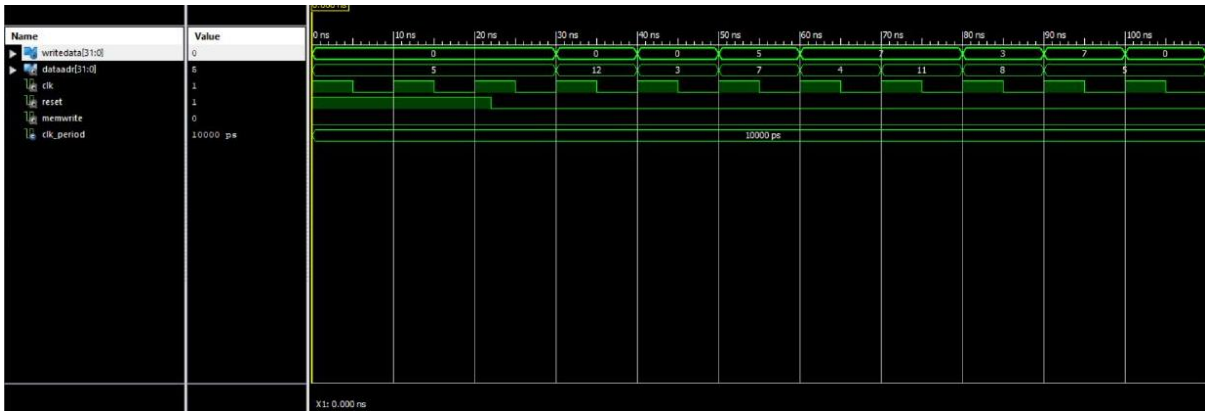

Figure 43:main module test bench simulation

Although our implementation was correct bas the test bench results didn't come out as expected.

<u>The Expected Results:</u>

INSTRUCTIONS 1-5: 1ST 5 CYCLES

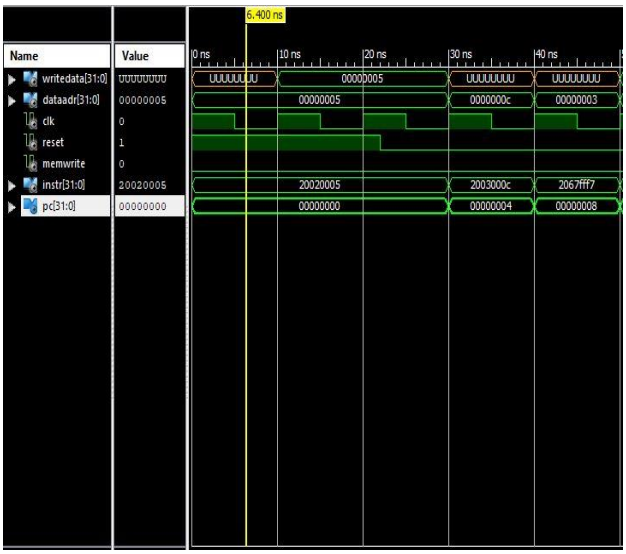

Figure 45: INSTRUCTIONS 1-5: 1ST 5 CYCLES

INSTRUCTIONS 6-10: 2ND 5 CYCLES

Figure 44: INSTRUCTIONS 6-10: 2ND 5 CYCLES

INSTRUCTIONS 11-15: 3RD 5 CYCLES LAST INSTRUCTION: CYCLE # 16

Figure 46: INSTRUCTIONS 11-15: 3RD 5 CYCLES

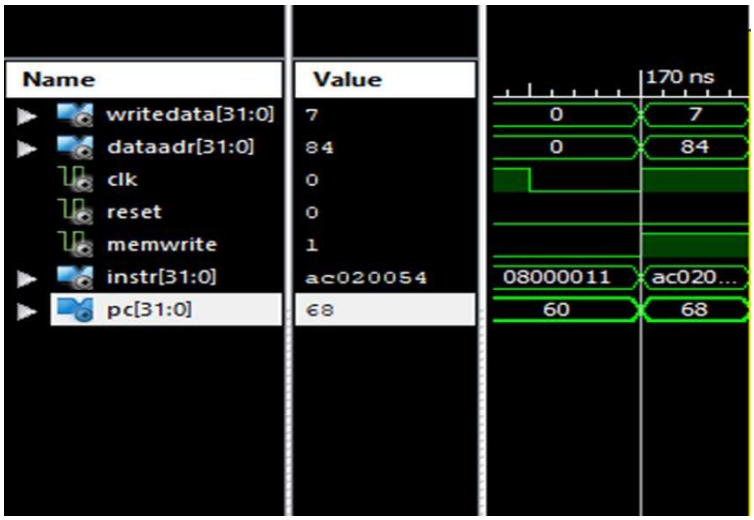

Figure 47: LAST INSTRUCTION: CYCLE # 16

Adder:

```
11  -- Description:
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use ieee.std_logic_unsigned.all;
23
24  -- Uncomment the following library declaration if using
25  -- arithmetic functions with Signed or Unsigned values
26  --use IEEE.NUMERIC_STD.ALL;
27
28  -- Uncomment the following library declaration if instantiating
29  -- any Xilinx primitives in this code.
30  --library UNISIM;
31  --use UNISIM.VComponents.all;
32
33  entity adder is
34  port( A : in std_logic_vector(31 downto 0);
35        B : in std_logic_vector(31 downto 0);
36        S : out std_logic_vector(31 downto 0));
37  end adder;
38
39  architecture Behavioral of adder is
40  begin
41
42  S <= A + B ;
43
44  end Behavioral;
45
```

Figure 48:adder code

Adder RTL schematic:

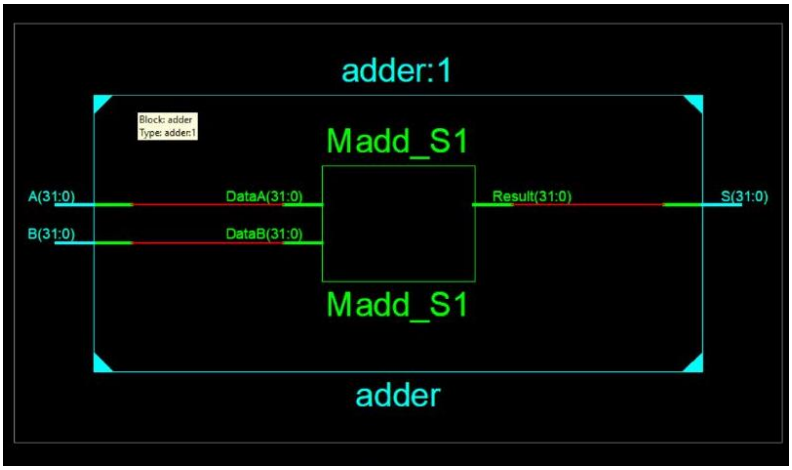

Figure 49:adder RTL schematic

Shifter:

```
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity Shifter is
33  Port ( INPUT : in  STD_LOGIC_VECTOR (31 downto 0);
34         OUTPUT : out  STD_LOGIC_VECTOR (31 downto 0));
35  end Shifter;
36
37  architecture Behavioral of Shifter is
38
39  begin
40
41  OUTPUT <= INPUT( 29 downto 0) & "00" ;
42
43  end Behavioral;
44
45
```

Figure 50:shifter code

Shifter RTL schematic:

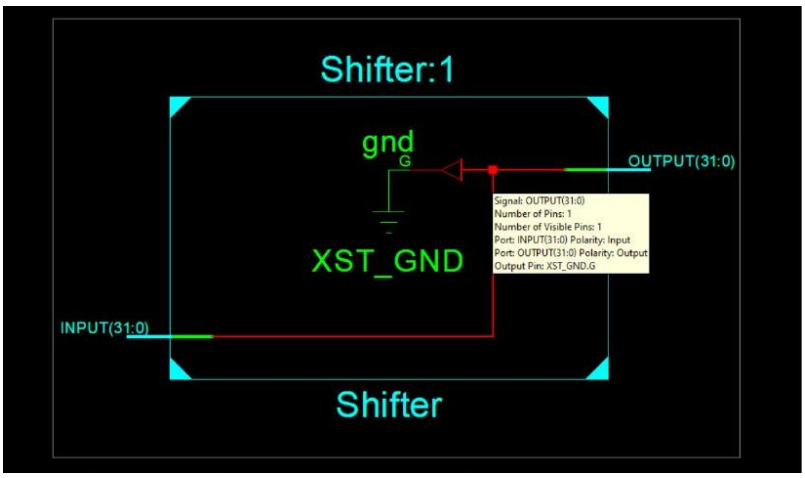

Figure 51:shifter RTL schematic

Sign extend:

```
12  --
13  -- Dependencies:
14  --
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity SignedBits is
33      Port ( INPUT : in  STD_LOGIC_VECTOR (15 downto 0);
34             OUTPUT : out  STD_LOGIC_VECTOR (31 downto 0));
35  end SignedBits;
36
37  architecture Behavioral of SignedBits is
38
39  begin
40
41  OUTPUT <= X"ffff"& INPUT WHEN INPUT(15)='1' ELSE
42  X"0000" & INPUT;
43
44
45
46  end Behavioral;
47
```

Figure 52:sign extend code

Sign extend RTL schematic:

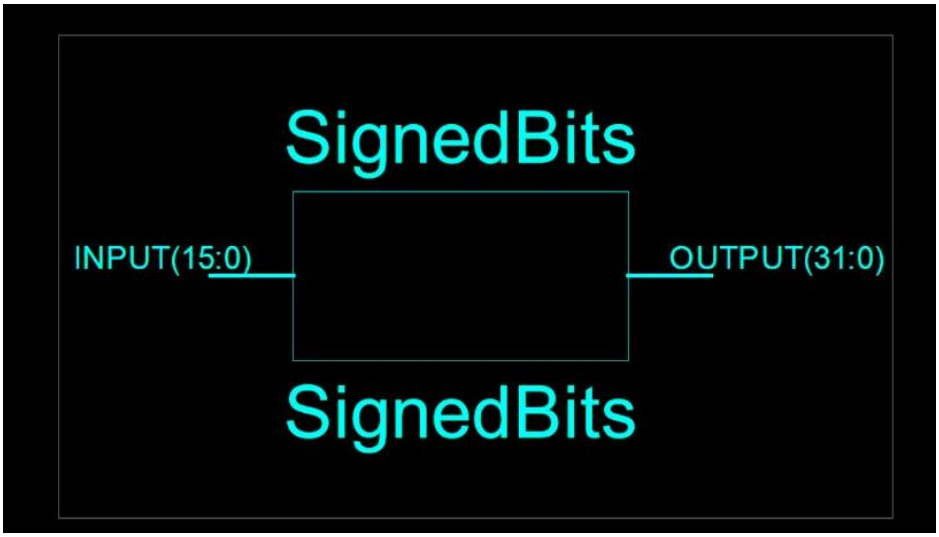

Figure 53:sign extend RTL schematic

MUX 5 bits:

```
15  -- Revision:
16  -- Revision 0.01 - File Created
17  -- Additional Comments:
18  --
19  ----------------------------------------------------------------------------------
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22
23  -- Uncomment the following library declaration if using
24  -- arithmetic functions with Signed or Unsigned values
25  --use IEEE.NUMERIC_STD.ALL;
26
27  -- Uncomment the following library declaration if instantiating
28  -- any Xilinx primitives in this code.
29  --library UNISIM;
30  --use UNISIM.VComponents.all;
31
32  entity MUX2x15bits is
33
34  port( A0 : in std_logic_vector(4 downto 0);
35        A1 : in std_logic_vector (4 downto 0);
36        Z1 : out std_logic_vector(4 downto 0);
37        S  : in std_logic);
38  end MUX2x15bits;
39
40  architecture Behavioral of MUX2x15bits is
41
42  begin
43
44  Z1 <= A0 when S='0' else
45        A1 when S='1' ;
46
47
48  end Behavioral;
49
```

Figure 54:MUX code

MUX 5bits RTL schematic:

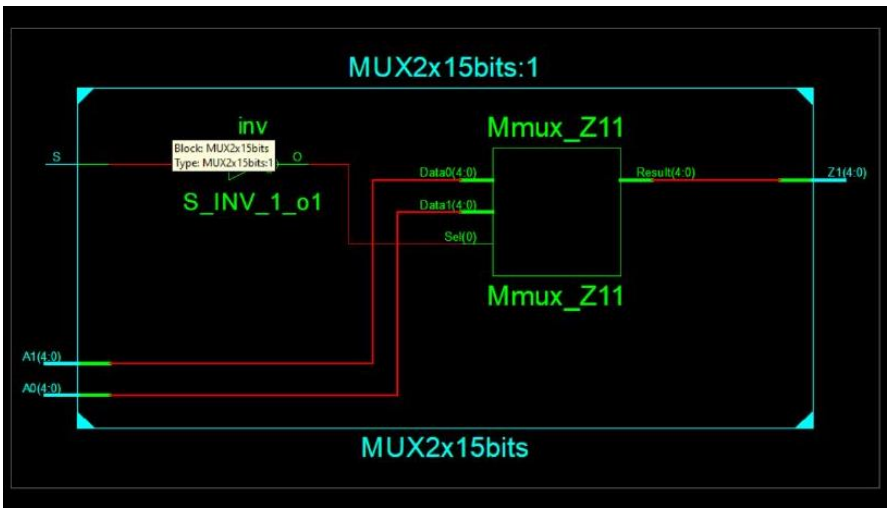

Figure 55:MUX RTL schematic

Instruction memory:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;
use ieee.numeric_std.all;
use IEEE.std_logic_textio.all;
library STD;
use STD.textio.all;
entity imem is -- instruction memory
port(a: in STD_LOGIC_VECTOR(5 downto 0);
rd: out STD_LOGIC_VECTOR(31 downto 0));
end;

architecture behave of imem is
begin
process is
file mem_file: TEXT;
variable L: line;
variable ch: character;
variable i, index, result: integer;
type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
variable mem: ramtype;
begin
-- initialize memory from file
for i in 0 to 63 loop -- set all contents low
mem(i) := (others => '0');
end loop;
index := 0;
FILE_OPEN (mem_file, "E:\SEM 4\Comp Arch\VHDL\memfile.dat", READ_MODE);
while not endfile(mem_file) loop
readline(mem_file, L);
result := 0;
for i in 1 to 8 loop
read (L, ch);
if '0' <= ch and ch <= '9' then
result := character'pos(ch) - character'pos('0');
elsif 'a' <= ch and ch <= 'f' then
result := character'pos(ch) - character'pos('a')+10;
else report "Format error on line" & integer'
image(index) severity error;
end if;
mem(index)(35-i*4 downto 32-i*4):= std_logic_vector(to_unsigned(result,4));
end loop;
index := index + 1;
end loop;
-- read memory
for i in 1 to 1000 loop
rd <= mem(CONV_INTEGER(a));
wait on a;
end loop;
end process;

end;
```

Figure 56:Imem code

Instruction memory RTL schematic:

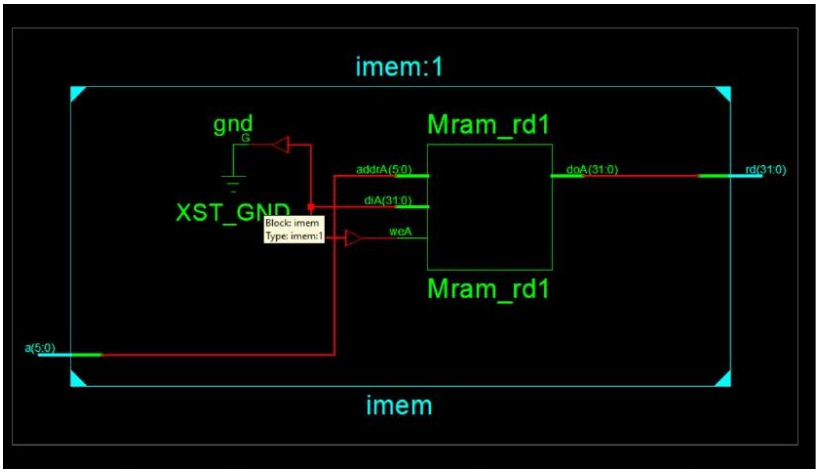

Figure 57:Imem RTL schematic

Data memory:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_SIGNED.all;
4  use IEEE.STD_LOGIC_ARITH.all;
5  use ieee.numeric_std.all;
6
7  entity dmem is -- data memory
8  port(clk, we: in STD_LOGIC;
9  a, wd: in STD_LOGIC_VECTOR (31 downto 0);
10 rd: out STD_LOGIC_VECTOR (31 downto 0));
11 end;
12 architecture behave of dmem is
13 begin
14 process  is
15 type ramtype is array (63 downto 0) of STD_LOGIC_VECTOR(31 downto 0);
16 variable mem: ramtype;
17 begin
18 -- read or write memory
19 for i in 1 to 1000 loop
20 if rising_edge(clk) then
21 if (we='1') then
22 mem (CONV_INTEGER('0'&a(7 downto 2))):= wd;
23 end if;
24 end if;
25 rd <= mem (CONV_INTEGER('0'&a (7 downto 2)));
26 wait on clk, a;
27 end loop;
28 end process;
29 end;
30
```

Figure 58:dmem code

Data memory RTL schematic:

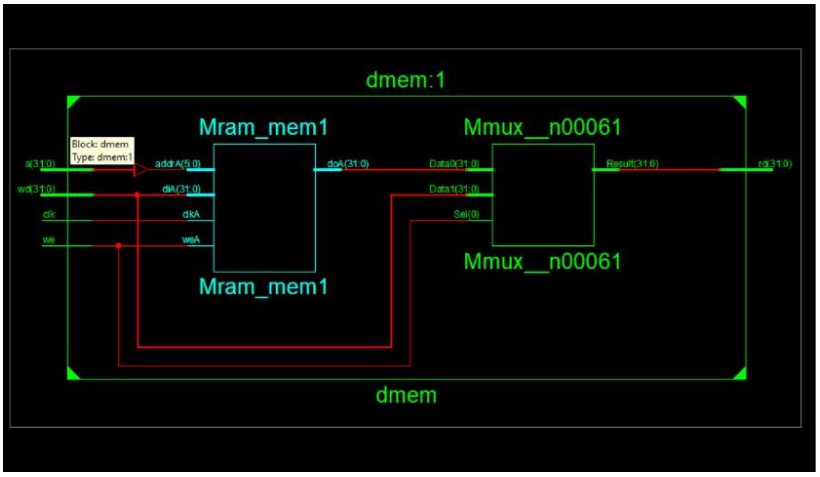

Figure 59:dmem RTL schematic

Code description:

- In phase 2 we implemented the data path consisting of:5 muxs, shifter,2 adders and sign extend.
- 1st adder used to increment the pc address by 4.
- 2nd adder used for branch instructions.
- The 1st mux we used was for write register according to the format R, and it will choose the instruction for [15 to 11], according to I format [25-21].
- 2nd mux we used was for the ALU's second input to choose between read data2 and 16-bit sign extend.
- 3rd mux was used to choose between read data from data memory and ALU output to write in the data of the register.
- 4th mux was used to choose between branch and pc next address.
- 5th mux was used to choose between branch mux output (4th mux) and the pc jump in case of the J format.
- Shifter was used for the immediate for I format.
- Sign extend used to make the 16 bits immediate to 32 bits for operations.
- ALU decoder is responsible for control of ALU operations according to its 2 bits.

Instruction	Aluop	funct	alucontrol
lw,sw,addi	00		0010
Beq	01		0110
R-type	10	100000	0010
		100010	0110
		100100	0000
		100101	0001
		101010	0111

Figure 60:ALU decoder table

- ALU decoder and the main decoder are components for the controller unit.

Instruction	op	Regwrite	Regdst	Alusrc	Branch	Mem write	Memtoreg	Jump	Aluop (1)	Aluop (0)
R-type	000000	1	1	0	0	0	0	0	1	0
Lw	100011	1	0	1	0	0	1	0	0	0
Sw	101011	0	0	1	0	1	0	0	0	0
beq	000100	0	0	0	1	0	0	0	0	1
addi	001000	1	0	1	0	0	0	0	0	0
J	000010	0	0	0	0	0	0	1	0	0

Figure 61:Main decoder table

Acknowledgement

Thanks to Dr Karim Emara for your effort through the semester and always providing feedback.

And, Thanks to the TAs Eng Aya and Eng Yasmin for helping us in the major task and their feedback and always being supportive and helpful.