الأكاديمية العربية للعلوم والتكنولوجيا والنقل البحري

Arab Academy for Science, Technology & Maritime Transport

MIPS

Name: Mahmoud Mostafa Mahmoud Ramadan
Reg No.: 19102461
Dr. Ahmed Abo El Farag
Eng. Esraa Khattab

Sources Needed for Mips

1) Program Counter

VHDL Module:

```
33  entity PCCounter is
34      Port ( Pc_In : in  STD_LOGIC_VECTOR (31 downto 0);
35           CLK : in  STD_LOGIC;
36             Pc_Out : out  STD_LOGIC_VECTOR (31 downto 0));
37  end PCCounter;
38
39  architecture Behavioral of PCCounter is
40  signal temp : STD_LOGIC_VECTOR (31 downto 0):= X"00000000";
41
42  begin
43  process(CLK, temp,Pc_In)
44  begin
45  Pc_Out<= temp;
46  if FALLING_EDGE(CLK) THEN temp <= Pc_In;
47  END IF;
48  end process;
49
50
51  end Behavioral;
```

2) Instruction Memory

VHDL Module :

```
33  entity InstMem is
34      Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
35             Instruction : out  STD_LOGIC_VECTOR (31 downto 0);
36             CLK : in STD_LOGIC);
37  end InstMem;
38
39  architecture Behavioral of InstMem is
40  type INSTArray is array (0 to 31) of STD_LOGIC_VECTOR (7 downto 0);
41  signal IM : INSTArray:= (
42                           x"00",x"85",x"10",x"20",    --add $v0, $a0, $a1
43                           x"AC",x"02",x"00",x"08",    --sw $v0, 8($zero)
44                           x"8C",x"06",x"00",x"08",    --lw $a2, 8($zero)
45                           x"10",x"46",x"00",x"01",    --beq $v0, $a2, Good_Processor
46                           x"00",x"46",x"88",x"2A",    --slt $s1, $v0, $a2
47                           x"00",x"A4",x"88",x"22",    --Good_Processor: sub $s1, $a1, $a0
48                           x"00",x"00",x"00",x"00",
49                           x"00",x"00",x"00",x"00");
50
51  begin
52
53  Instruction(31 downto 24) <= IM(to_integer(unsigned(Address)));
54  Instruction(23 downto 16) <= IM(to_integer(unsigned(Address))+1);
55  Instruction(15 downto 8) <= IM(to_integer(unsigned(Address))+2);
56  Instruction(7 downto 0) <= IM(to_integer(unsigned(Address))+3);
57
58  end Behavioral;
```

3) Register File

VHDL Module:

```
32  entity RegisterFile is
33      Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
34             ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
35             WriteReg : in  STD_LOGIC_VECTOR (4 downto 0);
36             WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
37             RegWrite : in  STD_LOGIC;
38             ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
39             ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
40             CLK : in  STD_LOGIC);
41
42  end RegisterFile;
```

```
44  architecture Behavioral of RegisterFile is
45  type RegtypeFile is array (0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
46  signal RegArray : RegtypeFile:= (X"00000000",X"00000000",
47                                   X"00000000",X"00000000",
48                                   X"00000005",X"00000007",
49                                   X"00000000",X"00000000",
50                                   X"00000000",X"00000000",
51                                   X"00000000",X"00000000",
52                                   X"00000000",X"00000000",
53                                   X"00000000",X"00000000",
54                                   X"00000000",X"00000000",
55                                   X"00000000",X"00000000",
56                                   X"00000000",X"00000000",
57                                   X"00000000",X"00000000",
58                                   X"00000000",X"00000000",
59                                   X"00000000",X"00000000",
60                                   X"00000000",X"00000000",
61                                   X"00000000",X"00000000");
62                                   --ThirdLine $a0,$a1
63  begin
64  Process(RegWrite,CLK, ReadReg1,ReadReg2,WriteReg,WriteData,RegWrite)
65  begin
66  ReadData1 <= RegArray(to_integer(unsigned(ReadReg1)));
67  ReadData2 <= RegArray(to_integer(unsigned(ReadReg2)));
68  if (rising_edge(CLK) and RegWrite = '1')
69  then
70  RegArray(to_integer(unsigned(WriteReg)))<= WriteData ;
71  end if;
72  end process;
73  end Behavioral;
```

4) ALU

VHDL Module:

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  
25  entity ALU4Bit is
26      Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
27             B : in  STD_LOGIC_VECTOR (31 downto 0);
28             ALUin : in  STD_LOGIC_VECTOR (3 downto 0);
29             zero : out  STD_LOGIC;
30             result : out  STD_LOGIC_VECTOR (31 downto 0));
31  end ALU4Bit;
35  begin
36  process(A,B,ALUin)
37  begin
38     if ALUin = "0000"
39        then result <= A and B ;
40     elsif ALUin = "0001"
41        then result <= A or B;
42     elsif ALUin = "0111"
43        then if  A < B then result <= X"00000001";
44        else result <= X"00000000";
45        end if;
46     elsif ALUin = "0010"
47        then result <= A + B;
48     elsif ALUin = "0110"
49        then result <= A - B;
50     elsif ALUin = "1100"
51        then result <= A nor B;
52     end if ;
53     if A = B
54     then zero <= '1';
55     else zero <= '0';
56     end if ;
57     end PROCESS;
58  
59  end Behavioral;
```

5) ALU Control

VHDL Module:

```
entity ALU_Control is
    Port ( ALUOP : in  STD_LOGIC_VECTOR (1 downto 0);
           FuncField : in  STD_LOGIC_VECTOR (5 downto 0);
           Operation : out  STD_LOGIC_VECTOR (3 downto 0));
end ALU_Control;

architecture Behavioral of ALU_Control is

begin
process(ALUOP,FuncField)
begin
if (ALUOP = "00") then Operation <= "0010";
elsif (ALUOP = "01") then Operation <= "0110";
elsif ((ALUOP = "10") and ((FuncField(3)='0')and(FuncField(2)='0') and (FuncField(1)='0')and (FuncField(0)='0'))) then Operation <= "0010";
elsif ((ALUOP = "10") and ((FuncField(3)='0')and(FuncField(2)='1') and (FuncField(1)='0')and (FuncField(0)='0'))) then Operation <= "0000";
elsif ((ALUOP = "10") and ((FuncField(3)='0')and(FuncField(2)='1') and (FuncField(1)='0')and (FuncField(0)='1'))) then Operation <= "0001";
elsif ((ALUOP(1) = '1') and ((FuncField(3)='0')and(FuncField(2)='0') and (FuncField(1)='1')and (FuncField(0)='0'))) then Operation <= "0110";
elsif ((ALUOP(1) = '1') and ((FuncField(3)='1')and(FuncField(2)='0') and (FuncField(1)='1')and (FuncField(0)='0'))) then Operation <= "0111";
end if ;
end process;
end Behavioral;
```

6) Control Unit

VHDL Module:

```
entity Control is
    Port ( OP : in  STD_LOGIC_VECTOR (5 downto 0);
           RegDst : out  STD_LOGIC;
           ALUSrc : out  STD_LOGIC;
           MemtoReg : out  STD_LOGIC;
           RegWrite : out  STD_LOGIC;
           MemRead : out  STD_LOGIC;
           MemWrite : out  STD_LOGIC;
           Branch : out  STD_LOGIC;
           ALUOP0 : out  STD_LOGIC;
           ALUOP1 : out  STD_LOGIC);
end Control;

architecture Behavioral of Control is

begin
PROCESS(OP)
BEGIN
IF OP = "000000" THEN RegDst <= '1'; ALUSrc <= '0';MemtoReg <= '0';RegWrite <='1' ;MemRead <= '0' ;MemWrite <= '0' ;Branch <= '0' ;ALUOP0 <= '1';ALUOP1 <= '0';
ELSIF OP = "100011" THEN RegDst <= '0'; ALUSrc <= '1';MemtoReg <= '1';RegWrite <='1' ;MemRead <= '1' ;MemWrite <= '0' ;Branch <= '0' ;ALUOP0 <= '0';ALUOP1 <= '0';
ELSIF OP = "101011" THEN ALUSrc <= '1';RegWrite <='0' ;MemRead <= '0' ;MemWrite <= '1' ;Branch <= '0' ;ALUOP0 <= '0';ALUOP1 <= '0';
ELSIF OP = "000100" THEN ALUSrc <= '0';RegWrite <='0' ;MemRead <= '0' ;MemWrite <= '0' ;Branch <= '1' ;ALUOP0 <= '0';ALUOP1 <= '1';

END IF;
END PROCESS;
```

7) Memory Unit

VHDL Module:

```
32  entity MemoryUnit is
33      Port ( MemRead : in  STD_LOGIC;
34             MemWrite : in  STD_LOGIC;
35             Address : in  STD_LOGIC_VECTOR (31 downto 0);
36             DataWrite : in  STD_LOGIC_VECTOR (31 downto 0);
37             DataRead : out  STD_LOGIC_VECTOR (31 downto 0);
38             CLK : in  STD_LOGIC);
39  end MemoryUnit;
40
41  architecture Behavioral of MemoryUnit is
42  type MEMArray is array (0 to 35) of STD_LOGIC_VECTOR (7 downto 0);
43  signal RAM: MEMArray:= (x"AB",x"CD",x"EF",x"00",
44                          x"75",x"74",x"65",x"72",
45                          x"20",x"41",x"72",x"63",
46                          x"68",x"69",x"74",x"65",
47                          x"12",x"34",x"56",x"78",
48                          x"7F",x"7F",x"6D",x"6D",
49                          x"00",x"00",x"00",x"00",
50                          x"78",x"78",x"6A",x"6A",
51                          x"00",x"00",x"00",x"01");
53  begin
54  process(MemRead,MemWrite,Address,CLK)
55  begin
56  if (MemRead = '1' and MemWrite ='0')
57  then DataRead(31 downto 24) <= RAM(to_integer(unsigned(Address)));
58  DataRead(23 downto 16) <= RAM(to_integer(unsigned(Address)+1));
59  DataRead(15 downto 8) <= RAM(to_integer(unsigned(Address)+2));
60  DataRead(7 downto 0) <= RAM(to_integer(unsigned(Address)+3));
61  elsif (MemRead = '0' and MemWrite ='1' and rising_edge(CLK))
62  then RAM(to_integer(unsigned(Address))) <= DataWrite(31 downto 24);
63  RAM(to_integer(unsigned(Address)+1)) <= DataWrite(23 downto 16);
64  RAM(to_integer(unsigned(Address)+2)) <= DataWrite(15 downto 8);
65  RAM(to_integer(unsigned(Address)+3)) <= DataWrite(7 downto 0);
66  end if ;
67  end process;
68  end Behavioral;
```

8) Adders

ADDER for 32 bit inputs

VHDL Module:

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  |
25
26  entity Adder32Bit is
27      Port ( In1 : in  STD_LOGIC_VECTOR (31 downto 0);
28             In2 : in  STD_LOGIC_VECTOR (31 downto 0);
29             Sum : out  STD_LOGIC_VECTOR (31 downto 0));
30  end Adder32Bit;
31
32  architecture Behavioral of Adder32Bit is
33
34  begin
35  process(In1,In2)
36  begin
37  Sum <= In1 + In2;
38  end process;
39
40  end Behavioral;
```

ADDER for PC+4

VHDL Module:

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  USE IEEE.STD_LOGIC_ARITH.ALL;
23  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25  |
26
27  entity Adder4Bit is
28      Port ( INPUT1 : in  STD_LOGIC_VECTOR (31 downto 0);
29             OUTPUT1 : out  STD_LOGIC_VECTOR (31 downto 0));
30  end Adder4Bit;
31
32  architecture Behavioral of Adder4Bit is
33
34  begin
35  process(INPUT1)
36  begin
37  OUTPUT1 <= INPUT1 + X"00000004";
38  end process;
39
40  end Behavioral;
```

9) Muxes

2-1 32Bit Mux

VHDL Module:

```
32  entity Mux2_1 is
33      Port ( IN1 : in  STD_LOGIC_VECTOR (31 downto 0);
34             IN2 : in  STD_LOGIC_VECTOR (31 downto 0);
35             S : in  STD_LOGIC;
36             OUTMUX : out  STD_LOGIC_VECTOR (31 downto 0));
37  end Mux2_1;
38
39  architecture Behavioral of Mux2_1 is
40  begin
41  process (IN1,IN2,S)
42  begin
43  if (S = '0') then OUTMUX <= IN1;
44  elsif (S = '1') then OUTMUX <= IN2;
45  end if ;
46  end process;
47
48  end Behavioral;
```

2-1 5Bit Mux

VHDL Module:

```
32  entity MUX_5 is
33      Port ( A : in  STD_LOGIC_VECTOR (4 downto 0);
34             B : in  STD_LOGIC_VECTOR (4 downto 0);
35             OUTM : out  STD_LOGIC_VECTOR (4 downto 0);
36             S : in  STD_LOGIC);
37  end MUX_5;
38
39  architecture Behavioral of MUX_5 is
40
41  begin
42  process(A,B,S)
43  begin
44  if (S = '0') then OUTM <= A;
45  elsif (S = '1') then OUTM <= B;
46  end if ;
47  end process;
48
49
50  end Behavioral;
```

10) Shift Left

VHDL Module:

```
32  entity ShiftLeft is
33      Port ( Inshift : in  STD_LOGIC_VECTOR (31 downto 0);
34             Outshift : out  STD_LOGIC_VECTOR (31 downto 0));
35  end ShiftLeft;
36
37  architecture Behavioral of ShiftLeft is
38
39  begin
40
41  process(Inshift)
42  begin
43  Outshift(1 downto 0) <= "00";
44  Outshift(31 downto 2) <= Inshift(29 downto 0);
45  end process;
46
47  end Behavioral;
```

11) Sign Extend

VHDL Module:

```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.STD_LOGIC_ARITH.ALL;
23  use IEEE.STD_LOGIC_UNSIGNED.ALL;
24  entity SignExtend32Bit is
25      Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
26             B : out  STD_LOGIC_VECTOR (31 downto 0));
27  end SignExtend32Bit;
28
29  architecture Behavioral of SignExtend32Bit is
30
31  begin
32  process(A)
33  begin
34  if (A(15)='0') then B <= X"00000000" + A;
35  elsif (A(15)='1')then B <= X"ffff0000" + A;
36  end if;
37  end process;
38
39  end Behavioral;
```

The Main Program Port Mapping and Components

```
32  entity PROJECT_MIPS is
33      Port ( CLK1 : in  STD_LOGIC);
34  end PROJECT_MIPS;
35
36  architecture Behavioral of PROJECT_MIPS is
37  --COMPONENTS
38
39  COMPONENT Adder32Bit is
40      Port ( In1 : in  STD_LOGIC_VECTOR (31 downto 0);
41             In2 : in  STD_LOGIC_VECTOR (31 downto 0);
42             Sum : out  STD_LOGIC_VECTOR (31 downto 0));
43  end component ;
44  COMPONENT MUX_5 is
45      Port ( A : in  STD_LOGIC_VECTOR (4 downto 0);
46             B : in  STD_LOGIC_VECTOR (4 downto 0);
47             OUTM : out  STD_LOGIC_VECTOR (4 downto 0);
48             S : in  STD_LOGIC);
49  end COMPONENT MUX_5;
50  component Adder4Bit is
51      Port ( INPUT1 : in  STD_LOGIC_VECTOR (31 downto 0);
52             OUTPUT1 : out  STD_LOGIC_VECTOR (31 downto 0));
53  end component;
54  component ALU4Bit is
55      Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
56             B : in  STD_LOGIC_VECTOR (31 downto 0);
57             ALUin : in  STD_LOGIC_VECTOR (3 downto 0);
58             zero : out  STD_LOGIC;
59             result : out  STD_LOGIC_VECTOR (31 downto 0));
60  end component;
61  component ALU_Control is
62      Port ( ALUOP : in  STD_LOGIC_VECTOR (1 downto 0);
63             FuncField : in  STD_LOGIC_VECTOR (5 downto 0);
64             Operation : out  STD_LOGIC_VECTOR (3 downto 0));
65  end component ;
66  component Control is
67      Port ( OP : in  STD_LOGIC_VECTOR (5 downto 0);
68             RegDst : out  STD_LOGIC;
69             ALUSrc : out  STD_LOGIC;
70             MemtoReg : out  STD_LOGIC;
71             RegWrite : out  STD_LOGIC;
72             MemRead : out  STD_LOGIC;
73             MemWrite : out  STD_LOGIC;
74             Branch : out  STD_LOGIC;
75             ALUOP0 : out  STD_LOGIC;
76             ALUOP1 : out  STD_LOGIC);
77  end component ;
78  component InstMem is
79      Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
80             Instruction : out  STD_LOGIC_VECTOR (31 downto 0);
81             CLK : in STD_LOGIC);
82  end component ;
83  component MemoryUnit is
84      Port ( MemRead : in  STD_LOGIC;
85             MemWrite : in  STD_LOGIC;
86             Address : in  STD_LOGIC_VECTOR (31 downto 0);
87             DataWrite : in  STD_LOGIC_VECTOR (31 downto 0);
88             DataRead : out  STD_LOGIC_VECTOR (31 downto 0);
89             CLK : in  STD_LOGIC);
90  end component;
91  component Mux2_1 is
92      Port ( IN1 : in  STD_LOGIC_VECTOR (31 downto 0);
93             IN2 : in  STD_LOGIC_VECTOR (31 downto 0);
94             S : in  STD_LOGIC;
95             OUTMUX : out  STD_LOGIC_VECTOR (31 downto 0));
96  end component;
```

```
component PCCounter is
    Port ( Pc_In : in  STD_LOGIC_VECTOR (31 downto 0);
         CLK : in  STD_LOGIC;
           Pc_Out : out  STD_LOGIC_VECTOR (31 downto 0));
end component;
component RegisterFile is
    Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
           ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteReg : in  STD_LOGIC_VECTOR (4 downto 0);
           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
           RegWrite : in  STD_LOGIC;
           ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
           ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
           CLK : in  STD_LOGIC);
end component;
component ShiftLeft is
    Port ( Inshift : in  STD_LOGIC_VECTOR (31 downto 0);
           Outshift : out  STD_LOGIC_VECTOR (31 downto 0));
end component;
component SignExtend32Bit is
    Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
           B : out  STD_LOGIC_VECTOR (31 downto 0));
end component ;
 --SIGNALS
 --PC Signals
 SIGNAL PC_OUT : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL PC_ADDER_OUT : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL PC_NEW_ADDRESS : STD_LOGIC_VECTOR (31 DOWNTO 0);
 --Instructions
 SIGNAL INSTRUCTION1: STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL OPCODE_CU : STD_LOGIC_VECTOR (5 DOWNTO 0);
 SIGNAL RS : STD_LOGIC_VECTOR (4 DOWNTO 0);
 SIGNAL RT : STD_LOGIC_VECTOR (4 DOWNTO 0);
 SIGNAL RD : STD_LOGIC_VECTOR (4 DOWNTO 0);
 SIGNAL FUNCTION_FIELD : STD_LOGIC_VECTOR (5 DOWNTO 0);
 SIGNAL INST15_0 : STD_LOGIC_VECTOR (15 DOWNTO 0);
 SIGNAL REGISTER_MUX : STD_LOGIC_VECTOR (4 DOWNTO 0);
 -- Regsiter outputs
 SIGNAL R1 : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL R2 : STD_LOGIC_VECTOR (31 DOWNTO 0);
 -- ALU IN and OUT
 SIGNAL IMMEDIATE : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL SHIFTLEFT_OUT : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL R2_IMMEDIATE_MUX : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL ALUCONTROL_OUT : STD_LOGIC_VECTOR (3 DOWNTO 0);
 SIGNAL ALU_RES_OUT : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL ZERO_CU : STD_LOGIC;
 --Branch
 SIGNAL NEW_BRANCH : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL BRANCH_CONTROLLER : STD_LOGIC;
 --Memory
 SIGNAL MEM_READ_OUT : STD_LOGIC_VECTOR (31 DOWNTO 0);
 SIGNAL WRITE_REG : STD_LOGIC_VECTOR (31 DOWNTO 0);
--Control Unit
SIGNAL REGDST_CU : STD_LOGIC;
SIGNAL BRANCH_CU : STD_LOGIC;
SIGNAL MEMREAD_CU : STD_LOGIC;
SIGNAL MEMTOREG_CU : STD_LOGIC;
SIGNAL MEMWRITE_CU : STD_LOGIC;
SIGNAL ALUSRC_CU : STD_LOGIC;
SIGNAL REGWRITE_CU : STD_LOGIC;
SIGNAL ALUOP_CU : STD_LOGIC_VECTOR (1 DOWNTO 0);

```

Port Mapping:

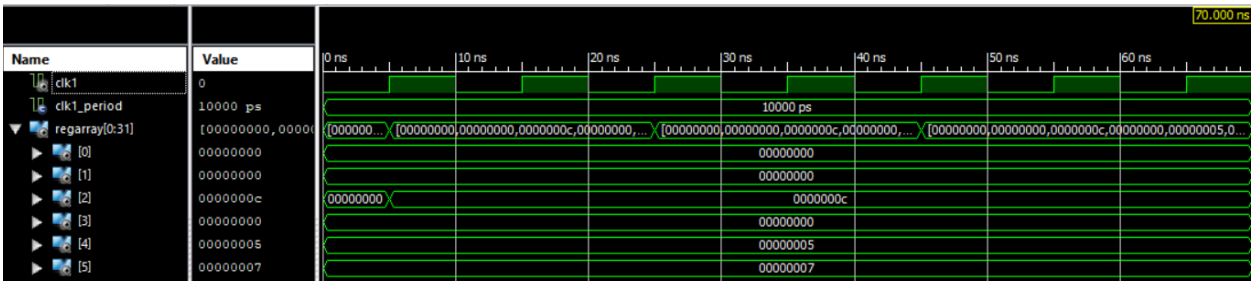
```
160  begin
161
162  --BREAKING DOWN INSTRUCTION(31 - 0)
163  OPCODE_CU <= INSTRUCTION1(31 DOWNTO 26);
164  RS <= INSTRUCTION1(25 DOWNTO 21);
165  RT <= INSTRUCTION1(20 DOWNTO 16);
166  RD <= INSTRUCTION1(15 DOWNTO 11);
167  INST15_0 <= INSTRUCTION1(15 DOWNTO 0);
168  FUNCTION_FIELD <= INSTRUCTION1(5 DOWNTO 0);
169  -- BRANCH_CONTROLLER
170  BRANCH_CONTROLLER <=  (BRANCH_CU AND ZERO_CU);
171  --CALCULATING NEW ADDRESS OF BRANCH
172  PC_BRANCH_ADDRESS : Adder32Bit
173      Port MAP( In1 => PC_ADDER_OUT,
174              In2 => SHIFTLEFT_OUT,
175              Sum => NEW_BRANCH);
176  --CALCULATING PC + 4
177  PC_NEXT_INSTRUCTION : Adder4Bit
178      Port MAP( INPUT1  => PC_OUT ,
179              OUTPUT1 => PC_ADDER_OUT );
180  --SELECTING THE NEW BRANCH OR PC+4
181  BRANCH_MUX : Mux2_1
182      Port MAP( IN1 => PC_ADDER_OUT,
183              IN2 => NEW_BRANCH,
184              S => BRANCH_CONTROLLER,
185              OUTMUX => PC_NEW_ADDRESS );
186
187  --FETCHING INSTRUCTION
188  INST_MEM : InstMem
189      Port MAP( Address => PC_OUT ,
190              Instruction => INSTRUCTION1,
191              CLK => CLK1);
192   --GETTING IMMEDIATE VALUE
193   SIGN_EXTEND : SignExtend32Bit
194       Port MAP( A => INST15_0 ,
195                 B => IMMEDIATE);
196   --SELECTING FOR WRITE REG
197   MUX_rEGISTER : MUX_5
198       Port MAP ( A => RT ,
199                 B => RD ,
200                 OUTM => REGISTER_MUX,
201                 S => REGDST_CU);
202   --GETTING REGISTERS
203   REGISTER_FILE : RegisterFile
204       Port MAP( ReadReg1 => RS,
205                 ReadReg2 => RT,
206                 WriteReg => REGISTER_MUX,
207                 WriteData => WRITE_REG,
208                 ReadData1 => R1,
209                 ReadData2 => R2,
210                 RegWrite => REGWRITE_CU,
211                 CLK => CLK1);
212   --GETTING OPERATIONS
213   ALUCONTROL : ALU_Control
214       Port MAP( ALUOP => ALUOP_CU ,
215                 FuncField => FUNCTION_FIELD ,
216                 Operation => ALUCONTROL_OUT );
217   --SELECTING R2 OR IMMEDIATE VALUE
218   ALU_MUX1 : Mux2_1
219       Port MAP( IN1 => R2,
220                 IN2 => IMMEDIATE,
221                 S => ALUSRC_CU,
222                 OUTMUX => R2_IMMEDIATE_MUX );
223
```

```
ALU : ALU4Bit
    Port MAP( A => R1,
           B => R2_IMMEDIATE_MUX,
           ALUin => ALUCONTROL_OUT ,
           zero => ZERO_CU,
           result => ALU_RES_OUT );
MEM : MemoryUnit
    Port MAP(
           MemRead => MEMREAD_CU,
           MemWrite => MEMWRITE_CU,
           Address => ALU_RES_OUT,
           DataWrite => R2,
           DataRead => MEM_READ_OUT,
           CLK => CLK1);

--SELECTING THE DATA TO BE WRITTEN IN THE REGISTER
MEM_MUX :  Mux2_1
    Port MAP( IN1 => ALU_RES_OUT,
           IN2 => MEM_READ_OUT,
           S => MEMTOREG_CU,
           OUTMUX => WRITE_REG);
--SETTING CONTROL UNIT SIGNALS
CONTROL_UNIT : Control
    Port MAP( OP => OPCODE_CU ,
           RegDst => REGDST_CU,
           ALUSrc => ALUSRC_CU,
           MemtoReg => MEMTOREG_CU,
           RegWrite => REGWRITE_CU,
           MemRead => MEMREAD_CU,
           MemWrite => MEMWRITE_CU,
           Branch => BRANCH_CU,
           ALUOP0 => ALUOP_CU(1),
           ALUOP1 => ALUOP_CU(0));

SHIFT_LEFT : ShiftLeft
    Port MAP( Inshift => IMMEDIATE ,
          Outshift => SHIFTLEFT_OUT );


PC_COUNT : PCCounter
    Port MAP( Pc_In => PC_NEW_ADDRESS,
         CLK => CLK1,
           Pc_Out => PC_OUT  );

end Behavioral;
```

Simulation of the Design:

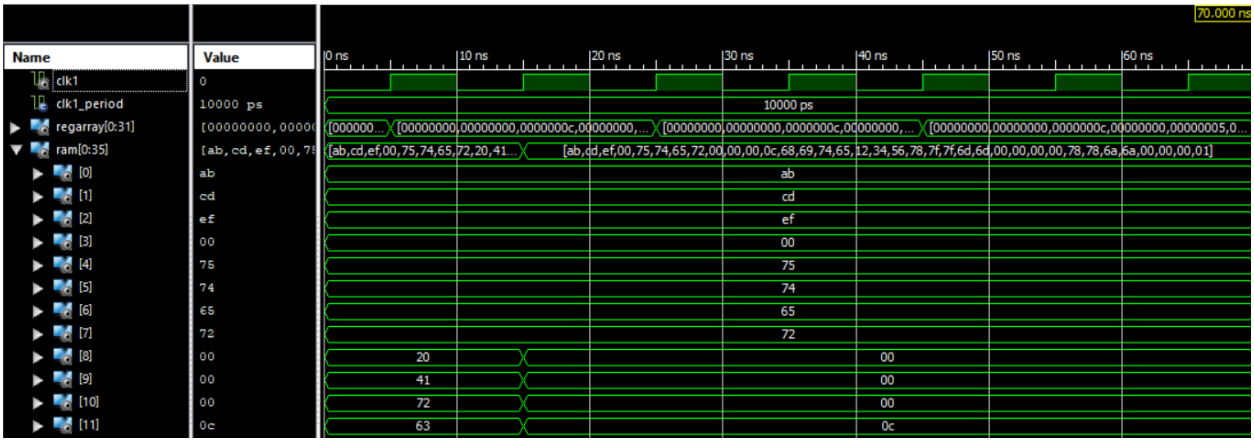
1- add $v0, $a0, $a1

This is R-Type Instruction so the operation is the addition of the values in registers $a0(4) and $a1(5) and save the result in $v0(2)
5+7 = x(c) as the simulation shows the operation is made correctly.



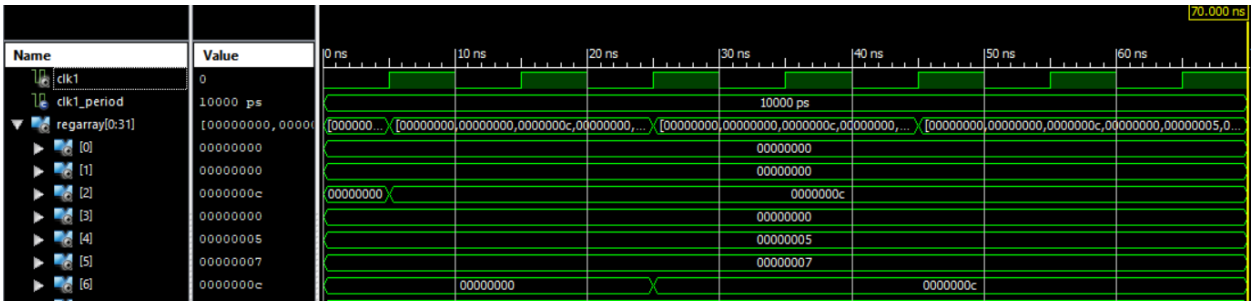
2- sw $v0, 8($zero)

This I-Type Instruction were the operation is done by calculating the address to which we will save the data of Register $v0 in the memory by the addition of the offset *4 with the reference address which is $zero (As we know from the previous instruction the value in $v0 is X”0000000c” from 8to 11)



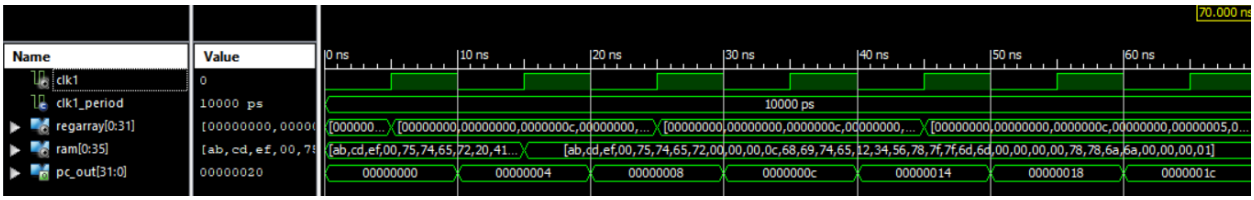
3- lw $a2, 8($zero)

This I-Type Instruction were the operation is done by reading the value stored in the memory at Address 8*4 +$zero and save it in the register $a2(6)



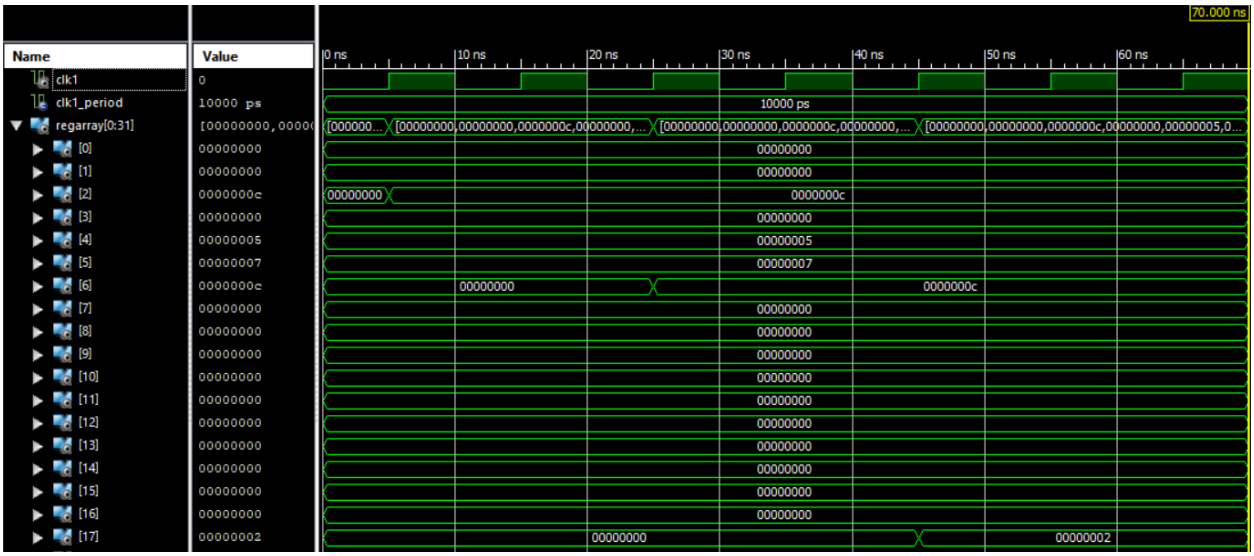
4- beq $v0, $a2, Good_Processor

This I-Type Instruction were the operation is done by subtracting the values stored in $v0(X”0000000c”) and $a2(X”0000000c”) if the result is zero then a new address is calculated by the addition of the immediate value which is sign extended and shifted to the left by 2 bits with the PC+4 signal calculated
(after the 4 clock the address after c is 10 but Because the condition was true the address changed to 14 which is the place where our label refers)



5- slt $s1, $v0, $a2

This is R-Type Instruction so the operation is the comparison of the values in registers $v0(2) and $a2(6) and if the value of $v0 is greater than $a2 then it will set $s1(17) to be equal 1
this instruction didn't work because the condition of beq was true and caused the sequence of the program to change



6- Good_Processor: sub $s1, $a1, $a0

This is R-Type Instruction so the operation is the Subtraction of the value in registers $a1(5) from $a0(4) and save the result in $s1(17)
7-5 = x(2) as the simulation shows the operation is made correctly