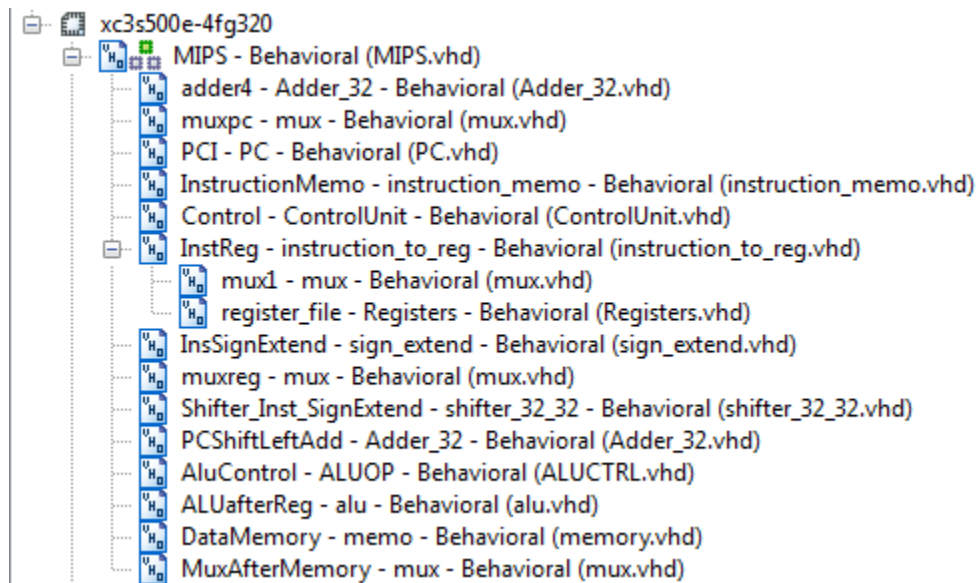


MIPS Single Cycle Project

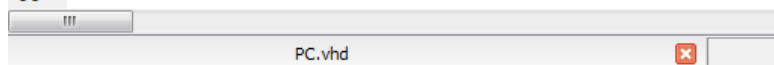
➤ VHDL Sources / Components:

- Hierarchy:



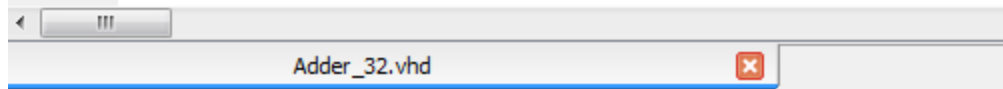
- Program Counter (PC):

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23
24 entity PC is
25     Port ( clk : in  STD_LOGIC;
26           inp : in  STD_LOGIC_VECTOR (31 downto 0);
27           outp : out STD_LOGIC_VECTOR (31 downto 0));
28 end PC;
29
30 architecture Behavioral of PC is
31     signal tmp : std_logic_vector (31 downto 0) := x"00000000";
32
33 begin
34     process(clk,inp,tmp) begin
35         if rising_edge(clk) then
36             outp <= tmp;
37         end if;
38         if falling_edge(clk) then
39             tmp<=inp;
40         end if;
41     end process;
42 end Behavioral;
```



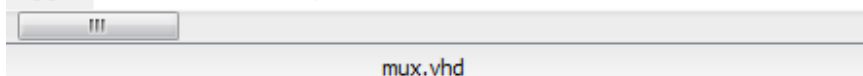
- Adder 32 bits:

```
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 use IEEE.NUMERIC_STD.ALL;
25
26 entity Adder_32 is
27     Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
28           B : in  STD_LOGIC_VECTOR (31 downto 0);
29           sum : out STD_LOGIC_VECTOR (31 downto 0));
30 end Adder_32;
31
32 architecture Behavioral of Adder_32 is
33
34 begin
35     sum <= A+B;
36
37 end Behavioral;
38
39
```



- Multiplexer (MUX):

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity mux is
24     generic(N: integer :=32);
25
26     port (
27         A:in std_logic_vector(N-1 downto 0);
28         B:in std_logic_vector(N-1 downto 0);
29         sel:in std_logic;
30         Y:out std_logic_vector(N-1 downto 0)
31     );
32
33 end mux;
34
35 architecture Behavioral of mux is
36
37 begin
38     process(A,B,sel)
39     begin
40         if sel ='0' then
41             Y<=A;
42         else
43             Y<=B;
44         end if;
45     end process;
46 end Behavioral;
```



- Shifter 32 bits (Shift left by 2)

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity shifter_32_32 is
24     Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
25           B : out  STD_LOGIC_VECTOR (31 downto 0));
26 end shifter_32_32;
27
28 architecture Behavioral of shifter_32_32 is
29
30 begin
31     B(31 downto 2) <= A(29 downto 0);
32     B(1 downto 0) <= "00";
33
34 end Behavioral;
35
```

!!!

shifter_32_32.vhd

- Instruction Memory:

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23
24 entity instruction_memo is
25     Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
26           Instruction : out  STD_LOGIC_VECTOR (31 downto 0));
27 end instruction_memo;
28
29 architecture Behavioral of instruction_memo is
30
31 type Memory is array(0 to 23) of STD_LOGIC_VECTOR(7 downto 0);
32 signal MM:Memory := (
33     X"00",X"85",X"10",X"20", -- add $v0, $a0, $a1
34     X"AC",X"02",X"00",X"08", -- sw $v0, 8($zero)
35     X"8C",X"06",X"00",X"08", -- lw $a2, 8($zero)
36     X"10",X"46",X"00",X"01", -- beq $v0, $a2, Good_Processor
37     X"00",X"46",X"88",X"2A", -- slt $s1, $v0, $a2
38     X"00",X"A4",X"88",X"22"); -- Good_Processor: sub $s1, $a1, $a0
39 begin
40     process(Address)
41     begin
42         Instruction(31 downto 24) <= MM(to_integer(unsigned(Address)));
43         Instruction(23 downto 16) <= MM(to_integer(unsigned(Address))+1);
44         Instruction(15 downto 8) <= MM(to_integer(unsigned(Address))+2);
45         Instruction(7 downto 0) <= MM(to_integer(unsigned(Address))+3);
46     end process;
47 end Behavioral;
```

!!!

instruction_memo.vhd

- Instruction to Registers

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity instruction_to_reg is
24     Port ( Instruction : in  STD_LOGIC_VECTOR (31 downto 0);
25           RegDst      : in  STD_LOGIC;
26           RegWrite    : in  STD_LOGIC;
27           WriteData    : in  STD_LOGIC_VECTOR (31 downto 0);
28           ReadData1    : out STD_LOGIC_VECTOR (31 downto 0);
29           ReadData2    : out STD_LOGIC_VECTOR (31 downto 0);
30           clk          : in  STD_LOGIC);
31 end instruction_to_reg;
32
33 architecture Behavioral of instruction_to_reg is
34     --components
35     component Registers is
36         Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
37               ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
38               WriteReg  : in  STD_LOGIC_VECTOR (4 downto 0);
39               WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
40               ReadData1 : out STD_LOGIC_VECTOR (31 downto 0);
41               ReadData2 : out STD_LOGIC_VECTOR (31 downto 0);
42               RegWrite  : in  STD_LOGIC;
43               CLK       : in  STD_LOGIC);
44     end component;
45
46     component mux is
47         generic(N: integer :=32);
```

```
46 component mux is
47 generic(N: integer :=32);
48
49 port( A:in std_logic_vector(N-1 downto 0);
50       B:in std_logic_vector(N-1 downto 0);
51       sel:in std_logic;
52       Y:out std_logic_vector(N-1 downto 0)
53 );
54 end component;
55
56 --signals
57 signal out_mux : STD_LOGIC_VECTOR(4 downto 0);
58
59 begin
60
61 --port mapping
62 mux1 : mux generic map(N=>5) port map(Instruction(20 downto 16),Instruction(15 downto 11), RegDst , out_mux);
63
64 register_file : Registers port map ( Instruction (25 downto 21),Instruction (20 downto 16),out_mux,WriteData,ReadData1,ReadData2,RegWrite,clk);
65
66 end Behavioral;
```

- Registers File

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23
24 entity Registers is
25     Port ( ReadReg1 : in  STD_LOGIC_VECTOR (4 downto 0);
26           ReadReg2 : in  STD_LOGIC_VECTOR (4 downto 0);
27           WriteReg  : in  STD_LOGIC_VECTOR (4 downto 0);
28           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
29           ReadData1 : out STD_LOGIC_VECTOR (31 downto 0);
30           ReadData2 : out STD_LOGIC_VECTOR (31 downto 0);
31           RegWrite  : in  STD_LOGIC;
32           CLK       : in  STD_LOGIC);
33 end Registers;
34
35 architecture Behavioral of Registers is
36     type regArray is array (0 to 31) of STD_LOGIC_VECTOR(31 downto 0);
37     signal regfile: regArray :=(
38     X"00000000", X"00000000", X"00000000", X"00000000",
39     X"00000005", X"00000007", X"00000000", X"00000000",
40     X"00000000", X"00000000", X"00000000", X"00000000",
41     X"00000000", X"00000000", X"00000000", X"00000000",
42     X"00000000", X"00000000", X"00000000", X"00000000",
43     X"00000000", X"00000000", X"00000000", X"00000000",
44     X"00000000", X"00000000", X"00000000", X"00000000",
45     X"00000000", X"00000000", X"00000000", X"00000000"
46     );
47 begin
48     ReadData1 <= regfile (TO_INTEGER (UNSIGNED (ReadReg1)));
49     ReadData2 <= regfile (TO_INTEGER (UNSIGNED (ReadReg2)));
50     PROCESS (WriteData, RegWrite, CLK)
51     BEGIN
52
53     IF RegWrite = '1' AND RISING_EDGE (CLK) THEN
54     regfile (TO_INTEGER (UNSIGNED (WriteReg))) <= WriteData;
55     END IF;
56     END PROCESS;
57 end Behavioral;
58
59
```

- Control Unit

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 entity ControlUnit is
23     Port ( Op : in  STD_LOGIC_VECTOR (5 downto 0);
24           RegDst : out  STD_LOGIC;
25           AluSrc : out  STD_LOGIC;
26           MemtoReg : out  STD_LOGIC;
27           RegWrite : out  STD_LOGIC;
28           MemRead : out  STD_LOGIC;
29           MemWrite : out  STD_LOGIC;
30           Branch : out  STD_LOGIC;
31           ALUOp1 : out  STD_LOGIC;
32           ALUOp0 : out  STD_LOGIC);
33 end ControlUnit;
34
35 architecture Behavioral of ControlUnit is
36
37 begin
38
39     process (Op)
40     begin
41         if Op ="000000" then --R-format
42             RegDst<='1';
43             AluSrc<='0';
44             MemtoReg <='0';
45             RegWrite <='1';
46             MemRead <='0';
47             MemWrite <='0';
48
49             Branch <='0';
50             ALUOp1 <='1';
51             ALUOp0 <='0';
52         elsif Op = "100011" then --lw
53             RegDst<='0';
54             AluSrc <='1';
55             MemtoReg<='1';
56             RegWrite <='1';
57             MemRead <='1';
58             MemWrite <='0';
```

```

58         Branch <='0';
59         ALUOp1 <='0';
60         ALUOp0 <='0';
61     elsif Op = "101011" then --sw
62         --RegDst<='0';
63         AluSrc <='1';
64         --MemtoReg<='1';
65         RegWrite <='0';
66         MemRead <='0';
67         MemWrite <='1';
68         Branch <='0';
69         ALUOp1 <='0';
70         ALUOp0 <='0';
71     elsif Op = "000100" then --beq
72         --RegDst<='0';
73         AluSrc <='0';
74         --MemtoReg<='1';
75         RegWrite <='0';
76         MemRead <='0';
77         MemWrite <='0';
78         Branch <='1';
79         ALUOp1 <='0';
80         ALUOp0 <='1';
81     end if;
82 end process;
83 end Behavioral;
84
85

```

ControlUnit.vhd

- Sign Extend 16 → 32 bits

```

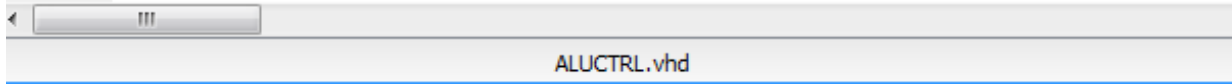
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity sign_extend is
24     Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
25           B : out  STD_LOGIC_VECTOR (31 downto 0));
26 end sign_extend;
27
28 architecture Behavioral of sign_extend is
29
30 begin
31     B(15 downto 0)<=A;
32     B(31 downto 16)<= (31 downto 16 => A(15));
33
34 end Behavioral;
35
36

```

sign_extend.vhd

- ALU Control

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 entity ALUOP is
24     Port ( func : in  STD_LOGIC_VECTOR (5 downto 0);
25           opcode : in  STD_LOGIC_VECTOR (1 downto 0);
26           op : out  STD_LOGIC_VECTOR (3 downto 0));
27 end ALUOP;
28
29 architecture Behavioral of ALUOP is
30
31 begin
32     process(opcode, func)
33     begin
34         if opcode = "00" then op<="0010"; -- load ad store
35         elsif opcode = "01" then op<="0110"; -- branch
36         elsif opcode = "10" and func = "100000" then op<="0010"; -- add
37         elsif opcode = "10" and func = "100010" then op<="0110"; -- sub
38         elsif opcode = "10" and func = "100100" then op<="0000"; -- and
39         elsif opcode = "10" and func = "100101" then op<="0001"; -- or
40         elsif opcode = "10" and func = "101010" then op<="0111"; -- set
41     end if;
42 end process;
43
44 end Behavioral;
45
```



- ALU

```
11 library IEEE;
12 use IEEE.STD_LOGIC_1164.ALL;
13 use IEEE.NUMERIC_STD.ALL;
14 use IEEE.STD_LOGIC_UNSIGNED.ALL;
15
16
17 entity alu is
18     Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
19           B : in  STD_LOGIC_VECTOR (31 downto 0);
20           ALU_CONTROL : in  STD_LOGIC_VECTOR (3 downto 0);
21           ALU_RESULT : out STD_LOGIC_VECTOR (31 downto 0);
22           ZERO : out  STD_LOGIC);
23 end alu;
24
25 architecture Behavioral of alu is
26     signal resultX: STD_LOGIC_VECTOR (31 downto 0);
27     begin
28     process(A,B,ALU_CONTROL)
29     begin
30
31     CASE ALU_CONTROL is
32     when "0000" => resultX <= A and B;
33     when "0001" => resultX <= A or B;
34     when "0010" => resultX <= A + B;
35     when "0110" => resultX <= A - B;
36
37     when "0111" =>
38     if A< B then resultX<=X"00000001";
39     else resultX<=X"00000000";
40     end if;
41
42     when "1100" => resultX<= A nor B;
43     when others => null;
44     resultX<=X"00000000";
45     end case;
46     end process;
47
48     zero <= '1' WHEN resultX=X"00000000" else '0';
49     ALU_RESULT<=resultX;
50 end Behavioral;
51
52
```

alu.vhd

- Data Memory (Main Memory)

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23
24 entity memo is
25     Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
26           WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
27           MemWrite : in  STD_LOGIC;
28           MemRead  : in  STD_LOGIC;
29           CLK       : in  STD_LOGIC;
30           ReadData  : out STD_LOGIC_VECTOR (31 downto 0));
31 end memo;
32
33 architecture Behavioral of memo is
34     type Memory is array(0 to 35) of STD_LOGIC_VECTOR(7 downto 0);
35     signal MM:Memory := (
36     X"AB",X"CD",X"EF",X"00",
37     X"75",X"74",X"65",X"72",
38     X"20",X"41",X"72",X"63",
39     X"68",X"69",X"74",X"65",
40     X"12",X"34",X"56",X"78",
41     X"7F",X"7F",X"6D",X"6D",
42     X"00",X"00",X"00",X"00",
43     X"78",X"78",X"6A",X"6A",
44     X"00",X"00",X"00",X"01");
45 begin
46     process(MemRead,MemWrite,Address,WriteData,CLK)
47     begin
48
49         if MemRead = '1' and MemWrite = '0' then
50             ReadData(31 downto 24) <= MM(to_integer(unsigned(Address)));
51             ReadData(23 downto 16) <= MM(to_integer(unsigned(Address))+1);
52             ReadData(15 downto 8)  <= MM(to_integer(unsigned(Address))+2);
53             ReadData(7  downto 0) <= MM(to_integer(unsigned(Address))+3);
54
55             elsif MemRead = '0' and MemWrite = '1' and rising_edge(CLK) then
56                 MM(to_integer(unsigned(Address))) <= WriteData(31 downto 24);
57                 MM(to_integer(unsigned(Address))+1) <= WriteData(23 downto 16);
58                 MM(to_integer(unsigned(Address))+2) <= WriteData(15 downto 8);
59                 MM(to_integer(unsigned(Address))+3) <= WriteData(7  downto 0);
60
61             end if;
62         end process;
63     end Behavioral;
64
65
```

III

memory.vhd

- MIPS

```
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22 use IEEE.NUMERIC_STD.ALL;
23 use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25 entity MIPS is
26     Port ( CLK : in  STD_LOGIC);
27 end MIPS;
28
29 architecture Behavioral of MIPS is
30
31     --PC
32     component PC is
33         Port ( clk : in  STD_LOGIC;
34               inp : in  STD_LOGIC_VECTOR (31 downto 0);
35               outp : out STD_LOGIC_VECTOR (31 downto 0));
36     end component;
37
38     --Instruction_to_Reg
39     component instruction_to_reg is
40         Port ( Instruction : in  STD_LOGIC_VECTOR (31 downto 0);
41               RegDst : in  STD_LOGIC;
42               RegWrite : in  STD_LOGIC;
43               WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
44               ReadData1 : out  STD_LOGIC_VECTOR (31 downto 0);
45               ReadData2 : out  STD_LOGIC_VECTOR (31 downto 0);
46               clk : in  STD_LOGIC);
47     end component;
48
49     --Adder32
50     component Adder_32 is
51         Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
52               B : in  STD_LOGIC_VECTOR (31 downto 0);
53               sum : out  STD_LOGIC_VECTOR (31 downto 0));
54     end component ;
55
56     --Control Unit
57     component ControlUnit is
58         Port ( Op : in  STD_LOGIC_VECTOR (5 downto 0);
59               RegDst : out  STD_LOGIC;
60               AluSrc : out  STD_LOGIC;
61               MemtoReg : out  STD_LOGIC;
62               RegWrite : out  STD_LOGIC;
63               MemRead : out  STD_LOGIC;
64               MemWrite : out  STD_LOGIC;
65               Branch : out  STD_LOGIC;
66               ALUOp1 : out  STD_LOGIC;
67               ALUOp0 : out  STD_LOGIC);
68     end component ;
69
70
71
```

```

72  --ALU
73  component alu is
74      Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
75            B : in  STD_LOGIC_VECTOR (31 downto 0);
76            ALU_CONTROL : in  STD_LOGIC_VECTOR (3 downto 0);
77            ALU_RESULT : out STD_LOGIC_VECTOR (31 downto 0);
78            ZERO : out  STD_LOGIC);
79  end component;
80
81  -- ALUControl
82  component ALUOP is
83      Port ( func : in  STD_LOGIC_VECTOR (5 downto 0);
84            opcode : in  STD_LOGIC_VECTOR (1 downto 0);
85            op : out  STD_LOGIC_VECTOR (3 downto 0));
86  end component;
87
88
89  -- Instruction memory
90  component instruction_memo is
91      Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
92            Instruction : out STD_LOGIC_VECTOR (31 downto 0));
93  end component;
94
95  --Data Memory
96  component memo is
97      Port ( Address : in  STD_LOGIC_VECTOR (31 downto 0);
98            WriteData : in  STD_LOGIC_VECTOR (31 downto 0);
99            MemWrite : in  STD_LOGIC;
100           MemRead : in  STD_LOGIC;
101           CLK : in  STD_LOGIC;
102           ReadData : out STD_LOGIC_VECTOR (31 downto 0));
103  end component;
104
105  --Shifter 26-28
106  component shifter_26_28 is
107      Port ( A : in  STD_LOGIC_VECTOR (25 downto 0);
108            B : out STD_LOGIC_VECTOR (27 downto 0));
109  end component ;
110
111  --Shifter 32-32
112  component shifter_32_32 is
113      Port ( A : in  STD_LOGIC_VECTOR (31 downto 0);
114            B : out STD_LOGIC_VECTOR (31 downto 0));
115  end component;
116
117  --Sign_extend
118  component sign_extend is
119      Port ( A : in  STD_LOGIC_VECTOR (15 downto 0);
120            B : out STD_LOGIC_VECTOR (31 downto 0));
121  end component;
122

```

```

123 --Mux
124 component mux is
125 generic(N: integer :=32);
126
127 port( A:in std_logic_vector(N-1 downto 0);
128       B:in std_logic_vector(N-1 downto 0);
129       sel:in std_logic;
130       Y:out std_logic_vector(N-1 downto 0)
131 );
132 end component;
133
134 --Signals Declaration
135 signal pc4 : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
136 signal BranchALUOutput : STD_LOGIC_VECTOR(31 downto 0);
137 signal pc_address_inp : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
138 signal pc_address_outp : STD_LOGIC_VECTOR(31 downto 0) := (others => '0');
139 signal BranchAndZero : STD_LOGIC ;
140 signal InstructionOutput : STD_LOGIC_VECTOR(31 downto 0);
141 signal RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,ALUOp1,ALUOp0 : STD_LOGIC;
142 signal WriteDatatoReg : STD_LOGIC_VECTOR(31 downto 0);
143 signal ReadData1,ReadData2 : STD_LOGIC_VECTOR (31 downto 0);
144 signal Ins_SignExtend_Out : STD_LOGIC_VECTOR (31 downto 0);
145 signal Mux_Reg_Out : STD_LOGIC_VECTOR (31 downto 0);
146 signal Shiftleft2ALU : STD_LOGIC_VECTOR (31 downto 0);
147 signal OperationtoALU: STD_LOGIC_VECTOR (3 downto 0);
148 signal ALUoutput : STD_LOGIC_VECTOR (1 downto 0);
149 signal ALUResult: STD_LOGIC_VECTOR (31 downto 0);
150 signal Zero :STD_LOGIC;
151 signal Branch :STD_LOGIC;
152 signal DataMemoryOutput :STD_LOGIC_VECTOR (31 downto 0); --
153
154

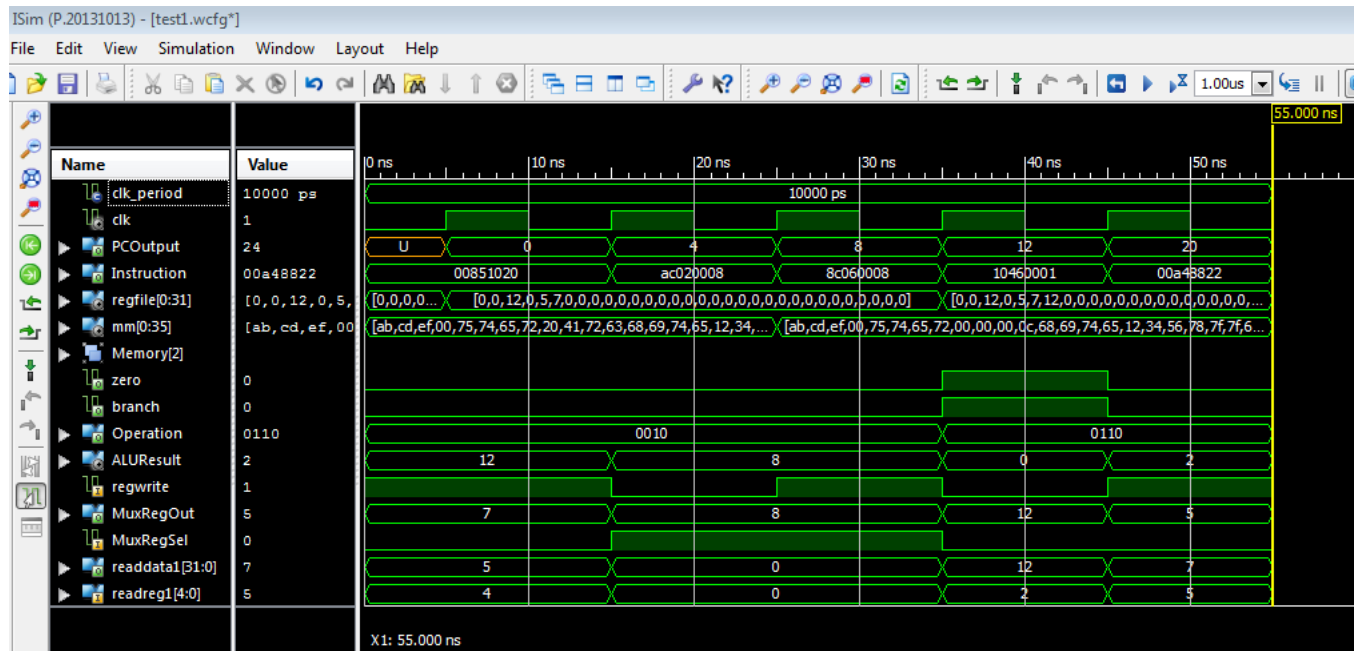
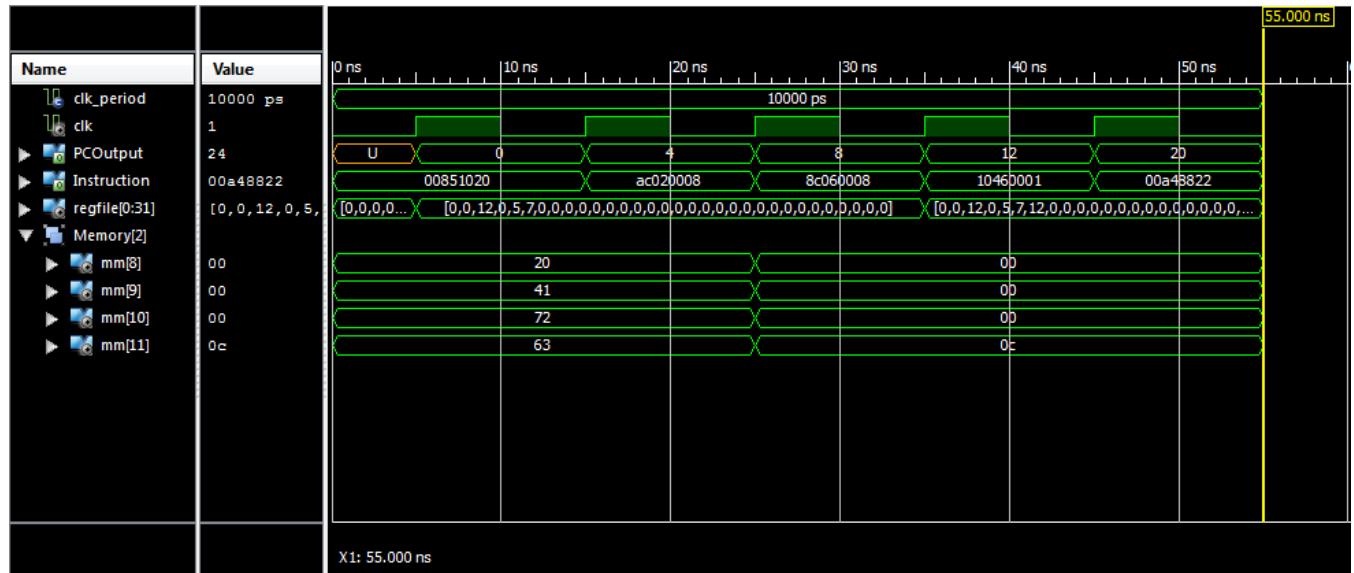
```

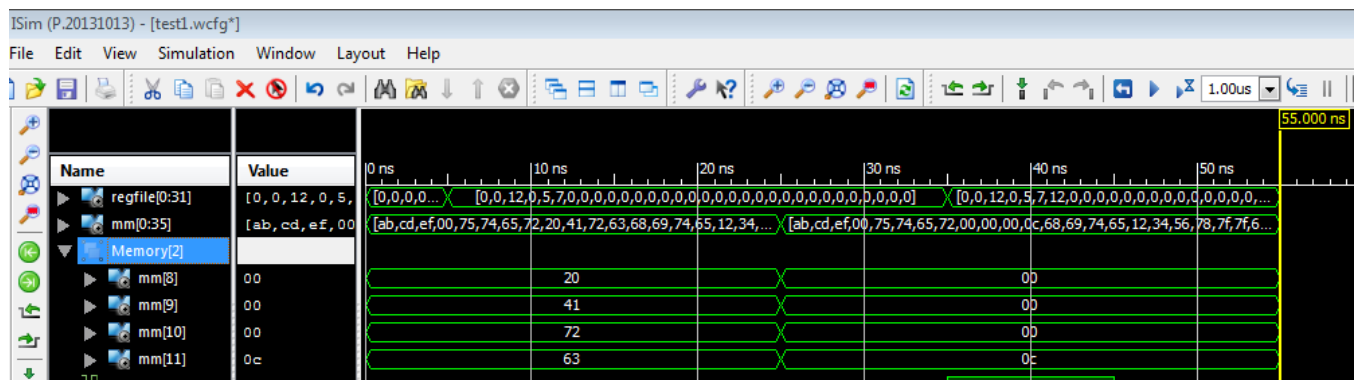
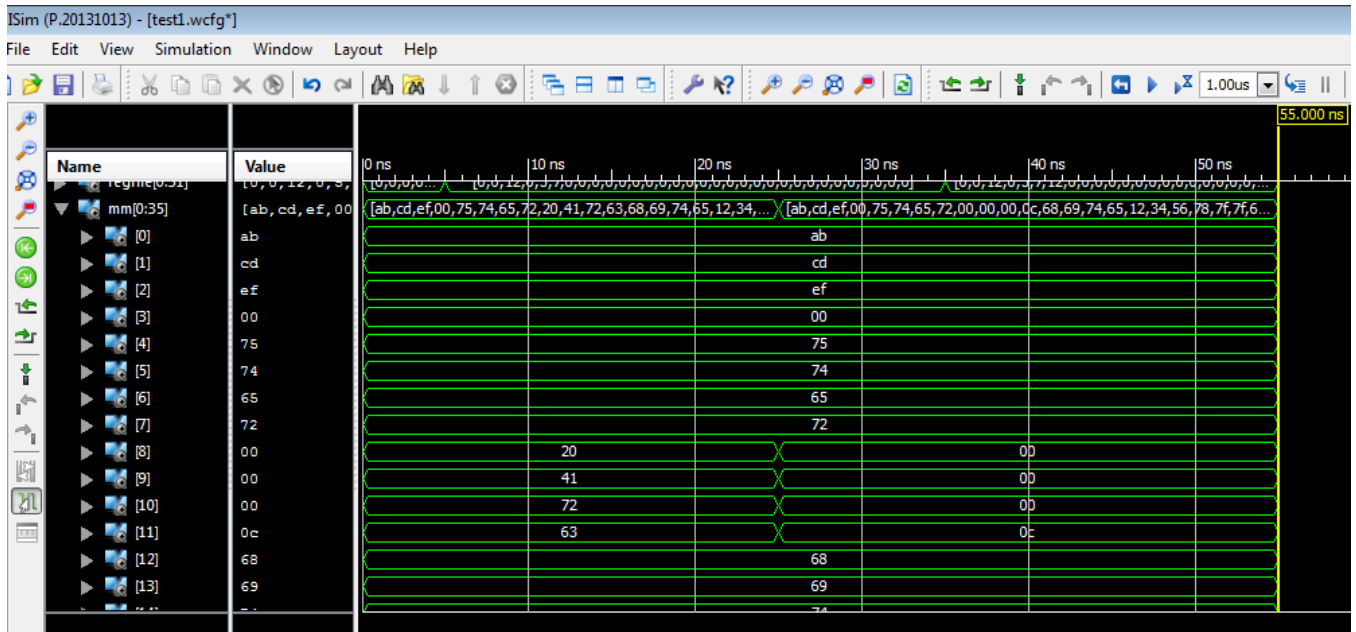
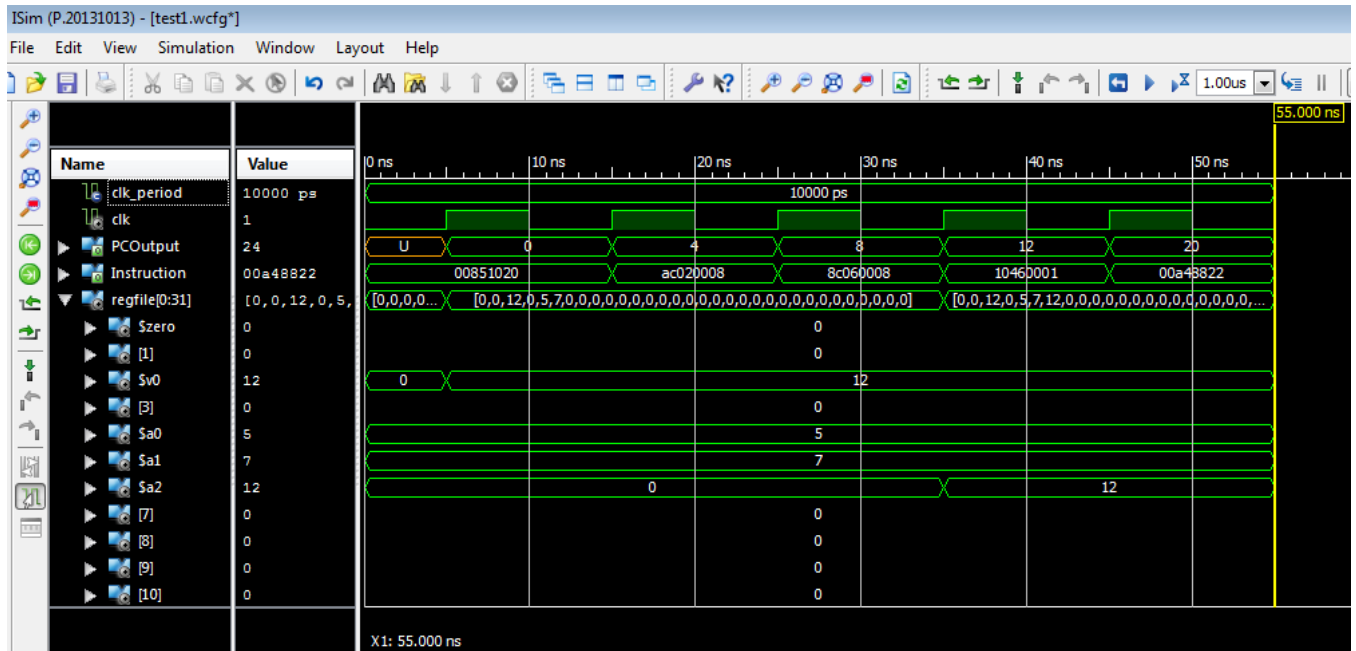
```

155 begin
156 -- port mapping
157 adder4 : Adder_32 port map(pc_address_outp,X"00000004",pc4);
158 muxpc: mux generic map (N=>32) port map (pc4,BranchALUOutput,BranchAndZero,pc_address_inp);
159 PCI : PC port map(CLK,pc_address_inp ,pc_address_outp);
160
161
162 InstructionMemo: instruction_memo port map(pc_address_outp,InstructionOutput);
163 Control: ControlUnit port map(InstructionOutput(31 downto 26),RegDst,AluSrc,MemtoReg,RegWrite,MemRead,MemWrite,Branch,ALUoutput(1),ALUoutput(0));
164 InstReg: instruction_to_reg port map(InstructionOutput,RegDst,RegWrite,WriteDatatoReg,ReadData1,ReadData2,CLK);
165
166 InsSignExtend: sign_extend port map(InstructionOutput(15 downto 0),Ins_SignExtend_Out);
167 muxreg: mux generic map (N=>32) port map (ReadData2,Ins_SignExtend_Out,AluSrc,Mux_Reg_Out);
168 Shifter_Inst_SignExtend : shifter_32_32 port map (Ins_SignExtend_Out,Shiftleft2ALU);
169 PCShiftLeftAdd : Adder_32 port map (pc4,Shiftleft2ALU,BranchALUOutput);
170
171 ALUControl : ALUOP port map(InstructionOutput(5 downto 0),ALUoutput,OperationtoALU);
172 ALUafterReg : alu port map (ReadData1,Mux_Reg_Out,OperationtoALU,ALUResult,Zero);
173 BranchAndZero <= Branch AND Zero;
174
175 DataMemory : memo port map (ALUResult,ReadData2,MemWrite,MemRead,CLK,DataMemoryOutput); --
176 MuxAfterMemory : mux generic map (N=>32) port map (ALUResult,DataMemoryOutput,MemtoReg,WriteDatatoReg); --
177 end Behavioral;
178
179

```

➤ Simulation Screenshots





➤ Tested Instructions:

Assembly Instruction	Object Code (Hex)
add \$v0, \$a0, \$a1	0x00851020
sw \$v0, 8(\$zero)	0xAC020008
lw \$a2, 8(\$zero)	0x8C060008
beq \$v0, \$a2, Good_Processor	0x10460001
slt \$s1, \$v0, \$a2	0x0046882A
Good_Processor: sub \$s1, \$a1, \$a0	0x00A48822