CSc 34300 - Computer Systems Design Laboratory

Instructor: Prof. Hamed Fazli

Lab **07**

Group **03**

Name
David Jimenez
Pablo Henriquez
Ratul Bin Rasul

Section **EF**

Date **December 15, 2022**

Important: Please follow the submission guidelines or your submission will be returned with zero point.

1. You are expected to submit two files (a PDF lab report and a ZIP file containing the source files) to Blackboard. Note that both files must be upload in the same submission attempt.
2. Each task must have its own source file(s). For VHDL assignments, all source files of this assignment must be in a single Xilinx VHDL project
3. For VHDL assignments, you must use the export function of the Xilinx ISE Design Suite to create the ZIP file properly. For detailed steps, please refer to the document "Exporting_VHDL_project_files" on Blackboard.
4. Your submission must be a PDF file. Naming convention:
   1) Report: "FirstName_LastName_Lab**XX**_CC**Y**.pdf"
   2) Source: "FirstName_LastName_Lab**XX**_CC**Y**.zip".
   *Please replace "XX" with the actual project number (2 digits), and "Y" with the section number, i.e. CC1 or CC2.
5. After the due day, all submissions are final. You cannot change it for any reasons. Double check before you make the submission.

CC-SiMP-32 Processor Top Module

The top module of the CC-SiMP-32 processor shown below is the main component that holds the rest of the components such as multiplexers, sign extensions, adders, shifter, PC, etc. as well as the defined interconnections between each of the components. The top module component takes in two signals called I_EN and I_CLK which represent the enable signal and clock of the processor respectively. When I_EN signal is '1', the processor will run otherwise the processor is stopped. When the program is finished, the processor is stopped using syscall. Since the processor is expected to only support the system call to terminate the program, this allows any syscall done to terminate the program. To validate our processor, a Fibonacci.asm program which computes the Fibonacci numbers and saves them in memory is used as our testing program.

CCSiMP32

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.NUMERIC_STD.ALL;
 4  library UNISIM;
 5  use UNISIM.VComponents.all;
 6
 7  entity CCSiMP32 is
 8      Port ( I_EN : in  STD_LOGIC;
 9             I_CLK : in  STD_LOGIC);
10  end CCSiMP32;
11
12  architecture CCSiMP32_Arch of CCSiMP32 is
13
14  component Acu is
15      Port ( I_ACU_ALUOp : in  STD_LOGIC_VECTOR (1 downto 0);
16             I_ACU_Funct : in  STD_LOGIC_VECTOR (5 downto 0);
17             O_ACU_CTL : out  STD_LOGIC_VECTOR (3 downto 0));
18  end component;
19
20  component adder32 is
21    port (
22      a      : in std_logic_vector(31 downto 0);
```

```
    b     : in std_logic_vector(31 downto 0);
    sum   : out std_logic_vector(31 downto 0);
    carry : out std_logic
  );
end component;

component ALU32 is
    Port ( I_EN : in STD_LOGIC;
           I_A : in  STD_LOGIC_VECTOR (31 downto 0);
           I_B : in  STD_LOGIC_VECTOR (31 downto 0);
           I_CTL : in  STD_LOGIC_VECTOR (3 downto 0);
           O_ZeroFlag : out  STD_LOGIC;
           O_Result : out  STD_LOGIC_VECTOR (31 downto 0));
end component;

component Dec is
    Port ( I_DEC_EN : in  STD_LOGIC;
           I_DEC_Opcode : in  STD_LOGIC_VECTOR (5 downto 0);
           O_DEC_RegDst : out  STD_LOGIC;
           O_DEC_Jump : out  STD_LOGIC;
           O_DEC_Beq : out  STD_LOGIC;
           O_DEC_Bne : out  STD_LOGIC;
           O_DEC_MemRead : out  STD_LOGIC;
           O_DEC_MemToReg : out  STD_LOGIC;
           O_DEC_ALUOp : out  STD_LOGIC_VECTOR (1 downto 0);
           O_DEC_MemWrite : out  STD_LOGIC;
           O_DEC_ALUSrc : out  STD_LOGIC;
           O_DEC_RegWrite : out  STD_LOGIC);
end component;

component FSM is
    Port ( I_FSM_CLK : in  STD_LOGIC;
           I_FSM_EN : in  STD_LOGIC;
           I_FSM_INST : in  STD_LOGIC_VECTOR (31 downto 0);
           O_IF : out  STD_LOGIC;
           O_ID : out  STD_LOGIC;
           O_EX : out  STD_LOGIC;
           O_ME : out  STD_LOGIC;
           O_WB : out  STD_LOGIC);
end component;

component MUX5 is
    Port ( I_Source_0 : in  STD_LOGIC_VECTOR (4 downto 0);
           I_Source_1 : in  STD_LOGIC_VECTOR (4 downto 0);
```

```
67              I_Mux_Sel : in  STD_LOGIC;
68              O_Mux : out  STD_LOGIC_VECTOR (4 downto 0));
69  end component;
70
71  component MUX32 is
72      Port ( I_Source_0 : in  STD_LOGIC_VECTOR (31 downto 0);
73             I_Source_1 : in  STD_LOGIC_VECTOR (31 downto 0);
74             I_Mux_Sel : in  STD_LOGIC;
75             O_Mux : out  STD_LOGIC_VECTOR (31 downto 0));
76  end component;
77
78  component PC is
79      Port ( I_PC : in  STD_LOGIC_VECTOR(31 DOWNTO 0);
80             I_PC_UPDATE : in  STD_LOGIC;
81             O_PC : out  STD_LOGIC_VECTOR(31 downto 0));
82  end component;
83
84  component PC_INC is
85      Port ( I_PC : in  STD_LOGIC_VECTOR(31 downto 0);
86             O_PC : out  STD_LOGIC_VECTOR(31 downto 0));
87  end component;
88
89  component RAM is
90      Port ( I_RAM_EN : in  STD_LOGIC;
91             I_RAM_RE : in  STD_LOGIC;
92             I_RAM_WE : in  STD_LOGIC;
93             I_RAM_ADDR : in  STD_LOGIC_VECTOR (31 downto 0);
94             I_RAM_DATA : in  STD_LOGIC_VECTOR (31 downto 0);
95             O_RAM_DATA : out  STD_LOGIC_VECTOR (31 downto 0));
96  end component;
97
98  component REG is
99      Port ( I_REG_EN : in  STD_LOGIC;
100            I_REG_WE : in  STD_LOGIC;
101            I_REG_SEL_RS : in  STD_LOGIC_VECTOR (4 downto 0);
102            I_REG_SEL_RT : in  STD_LOGIC_VECTOR (4 downto 0);
103            I_REG_SEL_RD : in  STD_LOGIC_VECTOR (4 downto 0);
104            I_REG_DATA_RD : in  STD_LOGIC_VECTOR (31 downto 0);
105            O_REG_DATA_A : out  STD_LOGIC_VECTOR (31 downto 0);
106            O_REG_DATA_B : out  STD_LOGIC_VECTOR (31 downto 0));
107 end component;
108
109 component ROM is
```

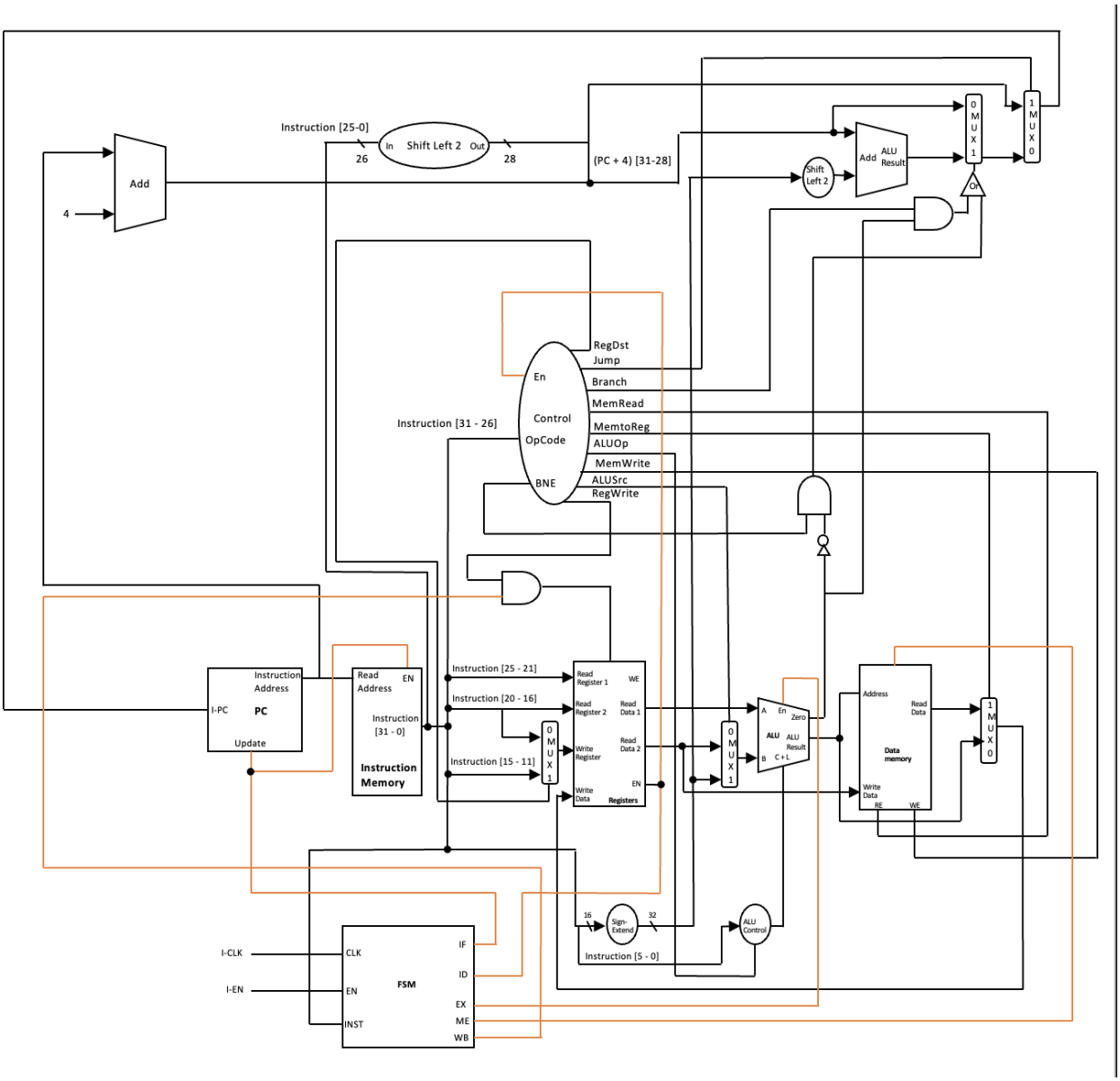
```
110     Port ( I_ROM_EN : in  STD_LOGIC;
111            I_ROM_ADDR : in  STD_LOGIC_VECTOR (31 downto 0);
112            O_ROM_DATA : out  STD_LOGIC_VECTOR (31 downto 0));
113 end component;
114
115 component SIGN_EXT is
116     Port ( I_EXT_16 : in  STD_LOGIC_VECTOR (15 downto 0);
117            O_EXT_32 : out  STD_LOGIC_VECTOR (31 downto 0));
118 end component;
119
120 component SL2 is
121 generic(
122 DEPTH : integer := 32;
123 Shift_Magnitude: integer := 2
124 );
125 port(
126    SI: in  std_logic_vector(31 downto 0);
127    SO: out std_logic_vector(31 downto 0));
128 end component;
129
130 component AND2 is port (I0: in std_logic; I1: in std_logic; O: out std_logic);
131 end component;
132
133 component OR2 is port (I0: in std_logic; I1: in std_logic; O: out std_logic);
134 end component;
135
136 component INV is port (I: in std_logic; O: out std_logic);
137 end component;
138
139 signal IF1, ID1, EX1, ME1, WB1, REG_DST, REGWRITE, WRITE_EN,
140        CARRYOUT1, BRA_CARRYOUT, ZERO, NOTZERO, ALUSRC, MEMREAD, MEMWRITE,
141        MEMTOREG, JUMP, BEQ, BNE, BNE_SELECT, BEQ_SELECT, BRA_SELECT: std_logic;
142 signal ALUOP: std_logic_vector(1 downto 0);
143 signal ACU_O: std_logic_vector(3 downto 0);
144 signal WRITE_REG: std_logic_vector(4 downto 0);
145 signal INST, INCREMENTED_PC, NEW_PC, CURRENT_PC, WRITE_TOREG, DATA_A, DATA_B,
146        DATA_RESULT, B_TOMUX, READ_TOMUX,
147        EXT_OUT, BRA_OFFSET, BRA_TOMUX, BRA_RESULT,
148        JTEMP, JA: STD_LOGIC_VECTOR(31 downto 0);
149
150 begin
151
152 U1: FSM port map(I_FSM_CLK => I_CLK, I_FSM_EN =>I_EN, I_FSM_INST=>INST,
153                  O_IF=>IF1, O_ID=>ID1, O_EX=>EX1, O_ME=>ME1, O_WB=>WB1);
```

```
154 U2: PC_INC port map(I_PC => CURRENT_PC, O_PC => INCREMENTED_PC);
155 U3: PC port map(I_PC=>NEW_PC, I_PC_UPDATE=>IF1, O_PC=>CURRENT_PC);
156 U4: ROM port map(I_ROM_EN =>IF1, I_ROM_ADDR => CURRENT_PC, O_ROM_DATA=>INST);
157 U5: AND2 port map(I0=>REGWRITE, I1=>WB1, O=>WRITE_EN);
158 U6: MUX5 port map(I_Source_0=>INST(20 downto 16), I_Source_1=>INST(15 downto 11),
159                   I_Mux_Sel=>REG_DST, O_Mux=>WRITE_REG); --0 = reg2, 1=writereg
160 U7: REG port map(I_REG_EN=>ID1, I_REG_WE=>WRITE_EN,
161                  I_REG_SEL_RS=>INST(25 downto 21),
162                  I_REG_SEL_RT=>INST(20 downto 16), I_REG_SEL_RD=>WRITE_REG,
163                  I_REG_DATA_RD=>WRITE_TOREG, O_REG_DATA_A=>DATA_A,
164                  O_REG_DATA_B=>B_TOMUX);
165 U8: MUX32 port map(I_Source_0=>B_TOMUX, I_Source_1=>EXT_OUT, I_Mux_Sel=>ALUSRC,
166                   O_Mux=>DATA_B);    --0 = extension, 1 = data2
167 U9: ALU32 port map(I_EN=>EX1, I_A=>DATA_A, I_B=>DATA_B, I_CTL=>ACU_O,
168     O_ZeroFlag=>ZERO, O_Result=>DATA_RESULT);--NEED ENABLE AND ZERO FLAG
169 U10: RAM port map(I_RAM_EN=>ME1,I_RAM_RE=>MEMREAD, I_RAM_WE=>MEMWRITE,
170                   I_RAM_ADDR=>DATA_RESULT, I_RAM_DATA=>B_TOMUX,
171                   O_RAM_DATA=>READ_TOMUX);
172 U11: MUX32 port map(I_Source_0=>DATA_RESULT, I_Source_1=>READ_TOMUX,
173                     I_Mux_Sel=>MEMTOREG, O_Mux=>WRITE_TOREG);
174 U12: SIGN_EXT port map(I_EXT_16=>INST(15 downto 0), O_EXT_32=>EXT_OUT);
175 U13: Acu port map(I_ACU_ALUOp=>ALUOP, I_ACU_Funct=>INST(5 downto 0),
176                   O_ACU_CTL=>ACU_O);
177 U14: Dec port map(I_DEC_EN=>ID1, I_DEC_Opcode=>INST(31 downto 26),
178                   O_DEC_RegDst=>REG_DST, O_DEC_Jump=>JUMP, O_DEC_Beq=>BEQ,
179                   O_DEC_Bne=>BNE,O_DEC_MemRead=>MEMREAD,
180                   O_DEC_MemToReg=>MEMTOREG, O_DEC_ALUOp=>ALUOP,
181                   O_DEC_MemWrite=>MEMWRITE, O_DEC_ALUSrc=>ALUSRC,
182                   O_DEC_RegWrite=>REGWRITE);
183 U15: INV port map(I=>ZERO, O=>NOTZERO);
184 U16: AND2 port map(I0=>BNE, I1=>NOTZERO, O=>BNE_SELECT);
185 U17: AND2 port map(I0=>BEQ, I1=>ZERO, O=>BEQ_SELECT);
186 U18: OR2 port map(I0=>BNE_SELECT, I1=>BEQ_SELECT, O=>BRA_SELECT);
187 U19: SL2 port map(SI=>EXT_OUT, SO=>BRA_OFFSET);
188 U20: adder32 port map(a=>INCREMENTED_PC, b=>BRA_OFFSET, sum=>BRA_TOMUX,
189                       carry=>BRA_CARRYOUT);
190 U21: MUX32 port map(I_Source_0=>INCREMENTED_PC, I_Source_1=>BRA_TOMUX,
191                     I_Mux_Sel=>BRA_SELECT, O_Mux=>BRA_RESULT);
192 U22: SL2 port map(SI=>INST, SO=>JTEMP);
193 JA <= INST(31 downto 28) & JTEMP(27 downto 0);
194 U23: MUX32 port map(I_Source_0=>BRA_RESULT, I_Source_1=>JA, I_Mux_Sel=>JUMP,
195                     O_Mux=>NEW_PC);
196 end CCSiMP32_Arch;
```

CCSimp32 Testbench

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY CCSiMP32_TestBench IS
END CCSiMP32_TestBench;

ARCHITECTURE behavior OF CCSiMP32_TestBench IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT CCSiMP32
    PORT(
         I_EN : IN  std_logic;
         I_CLK : IN  std_logic
        );
    END COMPONENT;


   --Inputs
   signal I_EN : std_logic := '0';
   signal I_CLK : std_logic := '0';

   -- Clock period definitions
    constant I_CLK_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: CCSiMP32 PORT MAP (
            I_EN => I_EN,
            I_CLK => I_CLK
          );

    -- Clock process definitions
    I_CLK_process :process
    begin
        I_CLK <= '0';
        wait for I_CLK_period/2;
        I_CLK <= '1';
        wait for I_CLK_period/2;
    end process;


    -- Stimulus process
    stim_proc: process
    begin

        -- insert stimulus here
        wait for 40 ns;
        I_EN <= '1';

        wait;
    end process;

END;
```

System Design Diagram

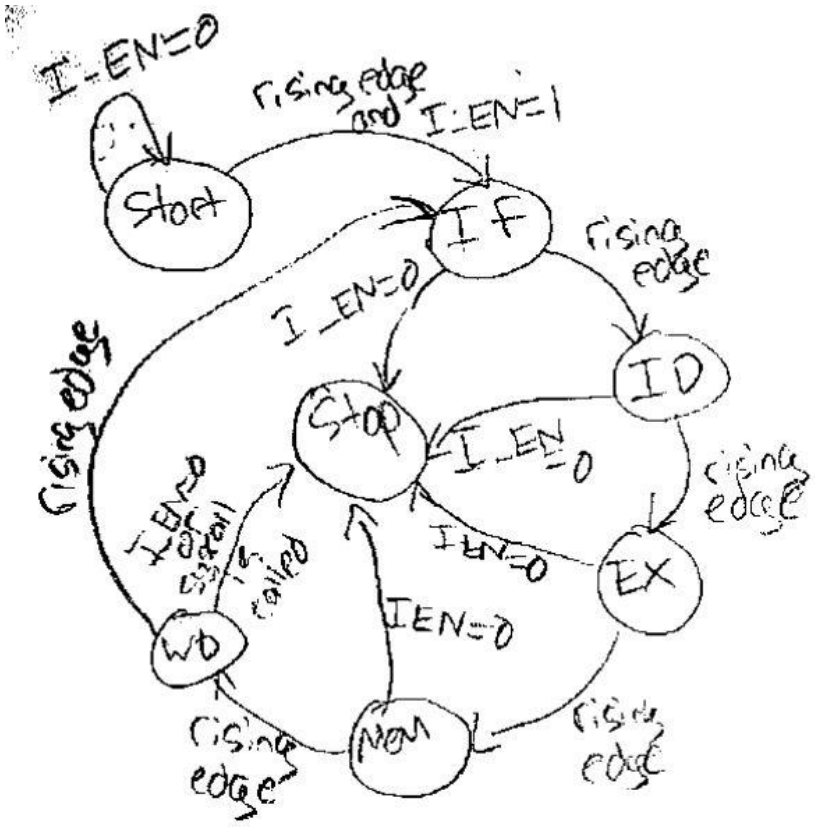


Finite State Machine (FSM)

For this simple MIPS processor program, the finite state machine focuses on going through the five stages of MIPS instruction execution with two additional stages consisting of a start stage for initialization and an end stage for stop. From the FSM diagram shown below, the program conducts an instruction fetch from memory (IF) from the initial start stage. The program afterwards conducts an instruction fetch from memory (IF) follows by decoding fetch instruction and reading registers (ID) to eventually execute operations or calculate addressing (EX). Next, we then access memory operand (MEM) and finally by writing the result back to the register

(WB). Each stage waits for their turn through the clock signal and terminates during the ID stage when the instruction is a syscall.

FSM Diagram



With an FSM diagram created, we can translate the diagram to an FSM VHDL implementation shown below where the FSM entity has three input signals (I_FSM_CLK, I_FSM_EN, I_FSM_INST) and five output signals (O_IF, O_ID, O_EX ,O_ME ,O_WB). At the initial stage, none of the signals are enabled until the FSM is enabled. Once the FSM is enabled, the next rising edge of the clock causes the FSM to move from the initial stage to the instruction fetch (IF) stage. During the instruction fetch (IF) stage, the FSM sends a signal to the PC to update the PC value and enables the ROM to begin fetching the updated instructions. After a single clock cycle, the FSM moves to the instruction decoding (ID) stage.

Once at the instruction decoding (ID) stage, the FSM sends a signal to the control (decoder module) to update the processor control's signal according to the OpCode of the instruction. A signal also gets sent to the register module to update the register outputs according to the inputs of RS and RT. After a single clock cycle, the FSM moves to the execution (EX) stage. At the execution stage, the FSM sends a signal to the ALU module which enables the inputs to update and in turn updates the outputs according to the inputs. After a single clock cycle, the FSM

moves to the MEM stage. During the MEM stage of the FSM, the FSM sends a signal to the RAM, which enables the address input to update and in turn update the data outputs. After a single clock cycle, the FSM moves to the WB stage. At the WB stage, the FSM sends a signal to an AND gate that takes a control reg write signal. Which means that only when the FSM is in the WB stage and the Write to register control signal is active, the register writes the register identified by the RD input. Once the WB stage is completed, the FSM can either move into the IF stage once more or move to the stop stage where all the outputs for the other stages (IF, ID, EX, MEM, WB) are disabled. Once the FSM reaches the stop stage, the FSM stays in the stop stage.

FSM VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6  use work.Common.all;
7
8  entity FSM is
9      Port ( I_FSM_CLK : in  STD_LOGIC;
10            I_FSM_EN : in  STD_LOGIC;
11            I_FSM_INST : in  STD_LOGIC_VECTOR (31 downto 0);
12            O_IF : out  STD_LOGIC;
13            O_ID : out  STD_LOGIC;
14            O_EX : out  STD_LOGIC;
15            O_ME : out  STD_LOGIC;
16            O_WB : out  STD_LOGIC);
17 end FSM;
18
19 architecture FSM_Arch of FSM is
20 signal state : CPU_states := FSM_INIT;
21 begin
22    process(I_FSM_CLK)
```

```
23    begin
24       if rising_edge(I_FSM_CLK) then
25          if I_FSM_EN = '1' then
26             if I_FSM_INST(5 downto 0) = "001100"
27             and I_FSM_INST(31 downto 26) = "000000"
28             and state = FSM_ID then --Check for program terminate syscall
29                state <= FSM_STOP;
30             else
31                if state = FSM_INIT then state <= FSM_IF;
32                elsif state = FSM_IF then state <= FSM_ID;
33                elsif state = FSM_ID then state <= FSM_EX;
34                elsif state = FSM_EX then state <= FSM_MEM;
35                elsif state = FSM_MEM then state <= FSM_WB;
36                elsif state = FSM_WB then state <= FSM_IF;
37                end if;
38             end if;
39          else
40             if state = FSM_INIT then state <= FSM_INIT;
41             else state <= FSM_STOP;
42             end if;
43          end if;
44       end if;
45    end process;
46    process(state)
47    begin
48       if state = FSM_IF then  --Instruction Fetch stage
49          O_IF <= '1';
50          O_ID <= '0';
51          O_EX <= '0';
52          O_ME <= '0';
53          O_WB <= '0';
54       elsif state = FSM_ID then  --Instruction Decode stage
55          O_IF <= '0';
56          O_ID <= '1';
57          O_EX <= '0';
58          O_ME <= '0';
59          O_WB <= '0';
60       elsif state = FSM_EX then  --Execution stage
61          O_IF <= '0';
62          O_ID <= '0';
63          O_EX <= '1';
64          O_ME <= '0';
65          O_WB <= '0';
66       elsif state = FSM_MEM then --Access Memory Stage
```

```
            O_IF <= '0';
            O_ID <= '0';
            O_EX <= '0';
            O_ME <= '1';
            O_WB <= '0';
        elsif state = FSM_WB then  --Write back stage
            O_IF <= '0';
            O_ID <= '0';
            O_EX <= '0';
            O_ME <= '0';
            O_WB <= '1';
        else  --INIT or STOP
            O_IF <= '0';
            O_ID <= '0';
            O_EX <= '0';
            O_ME <= '0';
            O_WB <= '0';
        end if;
    end process;
end FSM_Arch;
```

Package Common

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package Common is

type CPU_states is (FSM_INIT, FSM_IF, FSM_ID, FSM_EX, FSM_MEM, FSM_WB, FSM_STOP);

end Common;

package body Common is
end Common;
```

VHDL components for MIPS processor

ACU

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Acu is
    Port ( I_ACU_ALUOp : in  STD_LOGIC_VECTOR (1 downto 0);
           I_ACU_Funct : in  STD_LOGIC_VECTOR (5 downto 0);
           O_ACU_CTL : out  STD_LOGIC_VECTOR (3 downto 0));
end Acu;

architecture Acu_arch of Acu is

begin
process (I_ACU_ALUOp, I_ACU_Funct)
   begin
      -- I-type instruction:
      -- (lw, sw, addi , addiu)
      if (I_ACU_ALUOp = "00") then
         O_ACU_CTL <= "0010";
      -- this for beq and bne
      elsif (I_ACU_ALUOp = "01") then
         O_ACU_CTL <= "0110";
      -- r type instruction
      elsif (I_ACU_ALUOp = "10") then
         if (I_ACU_Funct = "100000") then --ADD
            O_ACU_CTL <= "0010";
         elsif (I_ACU_Funct = "100001") then --ADDU
            O_ACU_CTL <= "0010";
         elsif (I_ACU_Funct = "100010") then --SUBTRACT
            O_ACU_CTL <= "0110";
         elsif (I_ACU_Funct = "100100") then --AND
            O_ACU_CTL <= "0000";
         elsif (I_ACU_Funct = "100101") then --OR
            O_ACU_CTL <= "0001";
         elsif (I_ACU_Funct = "101010") then --SLT
            O_ACU_CTL <= "0111";
         end if;
      end if;
   end process;
end Acu_arch;
```

Adder 32

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_unsigned.all;
4  entity adder32 is
5    port (
6      a     : in std_logic_vector(31 downto 0);
7      b     : in std_logic_vector(31 downto 0);
8      sum   : out std_logic_vector(31 downto 0);
9      carry : out std_logic
10   );
11 end entity adder32;
12
13 architecture behavioural of adder32 is
14 signal temp : std_logic_vector(32 downto 0);
15 begin
16  process(a,b)
17  begin
18   temp <= ('0' & a) + ('0' & b);
19  end process;
20  sum  <= temp(31 downto 0);
21  carry   <= temp(32);
22 end behavioural;
```

Adder 32 Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY ADDER32_TestBench IS
6  END ADDER32_TestBench;
7
8  ARCHITECTURE behavior OF ADDER32_TestBench IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11     COMPONENT adder32
12     PORT(
13          a : IN  std_logic_vector(31 downto 0);
14          b : IN  std_logic_vector(31 downto 0);
15          sum : OUT  std_logic_vector(31 downto 0);
16          carry : OUT  std_logic
17         );
18     END COMPONENT;
19
20
21    --Inputs
22    signal a : std_logic_vector(31 downto 0) := (others => '0');
```

```
23    signal b : std_logic_vector(31 downto 0) := (others => '0');
24
25    --Outputs
26    signal sum : std_logic_vector(31 downto 0);
27    signal carry : std_logic;
28    -- No clocks detected in port list. Replace <clock> below with
29    -- appropriate port name
30
31
32  BEGIN
33
34    -- Instantiate the Unit Under Test (UUT)
35    uut: adder32 PORT MAP (
36           a => a,
37           b => b,
38           sum => sum,
39           carry => carry
40         );
41
42
43    -- Stimulus process
44    stim_proc: process
45     begin
46        -- hold reset state for 100 ns.
47  a <= "00000101111111110101010111111111";
48  b <= "00000000000000011010101011111111";
49  wait for 40 ns;
50  a <= "00000000000000000000000000000010";
51  b <= "00000000000000000000000000000011";
52
53        -- insert stimulus here
54
55        wait;
56     end process;
57
58  END;
```

ALU1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity ALU1 is
8      Port ( A : in  STD_LOGIC;
9             A_inv : in  STD_LOGIC;
10            B : in  STD_LOGIC;
11            B_inv : in  STD_LOGIC;
12            CarryIn : in  STD_LOGIC;
13            OP0 : in  STD_LOGIC;
14            OP1 : in  STD_LOGIC;
15            CarryOut : out  STD_LOGIC;
16            Result : out  STD_LOGIC);
17 end ALU1;
18
19 architecture alu1_arch of ALU1 is
20
21    component INV
22    port(I: in STD_LOGIC;
23    O: out STD_LOGIC);
24    end component;
25
26    component MUXCY
27    port(S,DI,CI: in STD_LOGIC;
28    O: out STD_Logic);
29    end component;
30
31    component AND2
32    port(I0,I1: in STD_LOGIC;
33    O: out STD_LOGIC);
34    end component;
35
36    component OR2
37    port(I0,I1: in STD_LOGIC;
38    O: out STD_LOGIC);
39    end component;
40
41    signal AN,BN,A0,B0,AND0,OR0,SUM: STD_LOGIC;
42
43 begin
44
```

```
45  INV_0: INV port map(A,AN);
46  INV_1: INV port map(B,BN);
47  MUXCY_A: MUXCY port map(A_inv,A,AN,A0);
48  MUXCY_B: MUXCY port map(B_inv,B,BN,B0);
49
50  AND_0: AND2 port map(A0,B0,AND0);
51  OR_0: OR2 port map(A0,B0,OR0);
52  ADD_0: entity work.full_adder1(adderF1_arch)
53     port map(A0,B0,CarryIn,CarryOut,SUM);
54
55  mux4_1: entity work.mux4_1(mux_arch)
56     port map(AND0,OR0,SUM,'-',OP0,OP1,Result);
57
58  end alu1_arch;
```

ALU1 Testbench

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY ALU1_testbench IS
6   END ALU1_testbench;
7
8   ARCHITECTURE behavior OF ALU1_testbench IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT ALU1
13      PORT(
14           A : IN  std_logic;
15           A_inv : IN  std_logic;
16           B : IN  std_logic;
17           B_inv : IN  std_logic;
18           CarryIn : IN  std_logic;
19           OP0 : IN  std_logic;
20           OP1 : IN  std_logic;
21           CarryOut : OUT  std_logic;
22           Result : OUT  std_logic
```

```
23          );
24      END COMPONENT;
25
26
27      --Inputs
28      signal A : std_logic := '0';
29      signal A_inv : std_logic := '0';
30      signal B : std_logic := '0';
31      signal B_inv : std_logic := '0';
32      signal CarryIn : std_logic := '0';
33      signal OP0 : std_logic := '0';
34      signal OP1 : std_logic := '0';
35
36      --Outputs
37      signal CarryOut : std_logic;
38      signal Result : std_logic;
39      -- No clocks detected in port list. Replace <clock> below with
40      -- appropriate port name
41
42  BEGIN
43
44      -- Instantiate the Unit Under Test (UUT)
45      uut: ALU1 PORT MAP (
46              A => A,
47              A_inv => A_inv,
48              B => B,
49              B_inv => B_inv,
50              CarryIn => CarryIn,
51              OP0 => OP0,
52              OP1 => OP1,
53              CarryOut => CarryOut,
54              Result => Result
55            );
56
57      -- Stimulus process
58      stim_proc: process
59      begin
60          -- A AND B
61          A_inv <= '0'; B_inv <= '0'; OP1 <= '0'; OP0 <= '0'; CarryIn <= '0';
62          A <= '0'; B <= '0'; wait for 10 ns;
63          A <= '0'; B <= '1'; wait for 10 ns;
64          A <= '1'; B <= '0'; wait for 10 ns;
65          A <= '1'; B <= '1'; wait for 40 ns;
66          -- A OR B
```

```
67        A_inv <= '0'; B_inv <= '0'; OP1 <= '0'; OP0 <= '1'; CarryIn <= '0';
68        A <= '0'; B <= '0'; wait for 10 ns;
69        A <= '0'; B <= '1'; wait for 10 ns;
70        A <= '1'; B <= '0'; wait for 10 ns;
71        A <= '1'; B <= '1'; wait for 40 ns;
72        -- A + B
73        -- CarryIn = '0'
74        A_inv <= '0'; B_inv <= '0'; OP1 <= '1'; OP0 <= '0'; CarryIn <= '0';
75        A <= '0'; B <= '0'; wait for 10 ns;
76        A <= '0'; B <= '1'; wait for 10 ns;
77        A <= '1'; B <= '0'; wait for 10 ns;
78        A <= '1'; B <= '1'; wait for 40 ns;
79        -- CarryIn = '1'
80        A_inv <= '0'; B_inv <= '0'; OP1 <= '1'; OP0 <= '0'; CarryIn <= '1';
81        A <= '0'; B <= '0'; wait for 10 ns;
82        A <= '0'; B <= '1'; wait for 10 ns;
83        A <= '1'; B <= '0'; wait for 10 ns;
84        A <= '1'; B <= '1'; wait for 40 ns;
85        -- A - B
86        A_inv <= '0'; B_inv <= '1'; OP1 <= '1'; OP0 <= '0'; CarryIn <= '1';
87        A <= '0'; B <= '0'; wait for 10 ns;
88        A <= '0'; B <= '1'; wait for 10 ns;
89        A <= '1'; B <= '0'; wait for 10 ns;
90        A <= '1'; B <= '1'; wait for 40 ns;
91
92        wait;
93     end process;
94
95  END;
```

ALU32

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4   library UNISIM;
5   use UNISIM.VComponents.all;
6
7   entity ALU32 is
8       Port ( I_EN : in STD_LOGIC;
9              I_A : in  STD_LOGIC_VECTOR (31 downto 0);
10             I_B : in  STD_LOGIC_VECTOR (31 downto 0);
11             I_CTL : in STD_LOGIC_VECTOR (3 downto 0);
12             O_ZeroFlag : out   STD_LOGIC;
13             O_Result : out  STD_LOGIC_VECTOR (31 downto 0));
14  end ALU32;
15
16  architecture alu32_arch of ALU32 is
17  begin
18  process(I_EN)
19  begin
20     if I_EN = '1' then
21        case I_CTL is
22           when "0000" => O_Result <= (I_A AND I_B);
```

```
        when "0001" => O_Result <= (I_A OR I_B);
        when "0010" => O_Result <= STD_LOGIC_VECTOR(signed(I_A)+signed(I_B));
        when "0110" => O_Result <= STD_LOGIC_VECTOR(signed(I_A)-signed(I_B));
        when others => O_Result <= "00000000000000000000000000000000";
    end case;
    if (signed(I_A)-signed(I_B)) = 0 then
        O_ZeroFlag <= '1';
    else O_ZeroFlag <= '0';
    end if;
  end if;
  end process;
end alu32_arch;
```

ALU32 Testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY ALU32_TestBench IS
END ALU32_TestBench;

ARCHITECTURE behavior OF ALU32_TestBench IS
    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT ALU32
    PORT(
         I_EN : IN  std_logic;
         I_A : IN  std_logic_vector(31 downto 0);
         I_B : IN  std_logic_vector(31 downto 0);
         I_CTL : IN  std_logic_vector(3 downto 0);
         O_ZeroFlag : OUT  std_logic;
         O_Result : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
```

```
23    signal I_EN : std_logic := '0';
24    signal I_A : std_logic_vector(31 downto 0) := (others => '0');
25    signal I_B : std_logic_vector(31 downto 0) := (others => '0');
26    signal I_CTL : std_logic_vector(3 downto 0) := (others => '0');
27
28    --Outputs
29    signal O_ZeroFlag : std_logic;
30    signal O_Result : std_logic_vector(31 downto 0);
31    -- No clocks detected in port list. Replace <clock> below with
32    -- appropriate port name
33
34
35 BEGIN
36
37    -- Instantiate the Unit Under Test (UUT)
38    uut: ALU32 PORT MAP (
39           I_EN => I_EN,
40           I_A => I_A,
41           I_B => I_B,
42           I_CTL => I_CTL,
43           O_ZeroFlag => O_ZeroFlag,
44           O_Result => O_Result
45         );
46
47
48    -- Stimulus process
49    stim_proc: process
50    begin
51       -- hold reset state for 100 ns.
52       wait for 100 ns;
53
54       I_EN <= '0';
55       I_A <= "00000000000000000000000000000100";
56       I_B <= "00000000000000000000000000000010";
57       I_CTL <= "0010";
58       wait for 20 ns;
59
60       I_EN <= '1';
61       I_A <= "00000000000000000000000000000100";
62       I_B <= "00000000000000000000000000000010";
63       I_CTL <= "0000";
64       wait for 20 ns;
65
66       I_EN <= '1';
```

```
        I_A <= "00000000000000000000000000000100";
        I_B <= "00000000000000000000000000000010";
        I_CTL <= "0001";
        wait for 20 ns;

        I_EN <= '1';
        I_A <= "00000000000000000000000000000100";
        I_B <= "00000000000000000000000000000010";
        I_CTL <= "0010";
        wait for 20 ns;

        I_EN <= '1';
        I_A <= "00000000000000000000000000000100";
        I_B <= "00000000000000000000000000000100";
        I_CTL <= "0110";
        wait for 20 ns;
        -- insert stimulus here

        wait;
    end process;
END;
```

Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity Dec is
    Port ( I_DEC_EN : in  STD_LOGIC;
           I_DEC_Opcode : in  STD_LOGIC_VECTOR (5 downto 0);
           O_DEC_RegDst : out  STD_LOGIC;
           O_DEC_Jump : out  STD_LOGIC;
           O_DEC_Beq : out  STD_LOGIC;
           O_DEC_Bne : out  STD_LOGIC;
           O_DEC_MemRead : out  STD_LOGIC;
           O_DEC_MemToReg : out  STD_LOGIC;
           O_DEC_ALUOp : out  STD_LOGIC_VECTOR (1 downto 0);
           O_DEC_MemWrite : out  STD_LOGIC;
           O_DEC_ALUSrc : out  STD_LOGIC;
           O_DEC_RegWrite : out  STD_LOGIC);
end Dec;

```

```
22  architecture Dec_arch of Dec is
23  begin
24     process(I_DEC_EN, I_DEC_Opcode)
25     begin
26        case (I_DEC_EN) is
27        when '1' =>
28           if I_DEC_Opcode = "000000" then -- addu
29              O_DEC_RegDst <= '1';
30              O_DEC_Jump <= '0';
31              O_DEC_Beq <= '0';
32              O_DEC_Bne <= '0';
33              O_DEC_MemRead <= '0';
34              O_DEC_MemtoReg <= '0';
35              O_DEC_ALUOp <= "10";
36              O_DEC_MemWrite <= '0';
37              O_DEC_ALUSrc <= '0';
38              O_DEC_RegWrite <= '1';
39           elsif (I_DEC_Opcode = "001000" OR I_DEC_Opcode = "001001") then -- addi
40              O_DEC_RegDst <= '0';
41              O_DEC_Jump <= '0';
42              O_DEC_Beq <= '0';
43              O_DEC_Bne <= '0';
44              O_DEC_MemRead <= '0';
45              O_DEC_MemtoReg <= '0';
46              O_DEC_ALUOp <= "00";
47              O_DEC_MemWrite <= '0';
48              O_DEC_ALUSrc <= '1';
49              O_DEC_RegWrite <= '1';
50           elsif I_DEC_Opcode = "000100" then -- beq
51              O_DEC_RegDst <= '0';
52              O_DEC_Jump <= '0';
53              O_DEC_Beq <= '1';
54              O_DEC_Bne <= '0';
55              O_DEC_MemRead <= '0';
56              O_DEC_MemtoReg <= '0';
57              O_DEC_ALUOp <= "01";
58              O_DEC_MemWrite <= '0';
59              O_DEC_ALUSrc <= '0';
60              O_DEC_RegWrite <= '0';
61           elsif I_DEC_Opcode = "000101" then -- bne
62              O_DEC_RegDst <= '0';
63              O_DEC_Jump <= '0';
64              O_DEC_Beq <= '0';
65              O_DEC_Bne <= '1';
```

```
66              O_DEC_MemRead <= '0';
67              O_DEC_MemtoReg <= '0';
68              O_DEC_ALUOp <= "01";
69              O_DEC_MemWrite <= '0';
70              O_DEC_ALUSrc <= '0';
71              O_DEC_RegWrite <= '0';
72           elsif I_DEC_Opcode = "101011" then -- sw
73              O_DEC_RegDst <= '0';
74              O_DEC_Jump <= '0';
75              O_DEC_Beq <= '0';
76              O_DEC_Bne <= '0';
77              O_DEC_MemRead <= '0';
78              O_DEC_MemtoReg <= '0';
79              O_DEC_ALUOp <= "00";
80              O_DEC_MemWrite <= '1';
81              O_DEC_ALUSrc <= '1';
82              O_DEC_RegWrite <= '0';
83           elsif I_DEC_Opcode = "100011" then -- lw
84              O_DEC_RegDst <= '0';
85              O_DEC_Jump <= '0';
86              O_DEC_Beq <= '0';
87              O_DEC_Bne <= '0';
88              O_DEC_MemRead <= '1';
89              O_DEC_MemtoReg <= '1';
90              O_DEC_ALUOp <= "00";
91              O_DEC_MemWrite <= '0';
92              O_DEC_ALUSrc <= '1';
93              O_DEC_RegWrite <= '1';
94           elsif I_DEC_Opcode = "000010" then -- j
95              O_DEC_RegDst <= '0';
96              O_DEC_Jump <= '1';
97              O_DEC_Beq <= '0';
98              O_DEC_Bne <= '0';
99              O_DEC_MemRead <= '0';
100             O_DEC_MemtoReg <= '0';
101             O_DEC_ALUOp <= "00";
102             O_DEC_MemWrite <= '0';
103             O_DEC_ALUSrc <= '0';
104             O_DEC_RegWrite <= '0';
105          else
106             NULL;
107          end if;
108       when others =>
109          NULL;
110    end case;
111 end process;
112 end Dec_arch;
```

Full Adder 1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity full_adder1 is
8      Port ( A : in  STD_LOGIC;
9             B : in  STD_LOGIC;
10            Cin : in  STD_LOGIC;
11            Cout : out  STD_LOGIC;
12            SUM : out  STD_LOGIC);
13 end full_adder1;
14
15 architecture adderF1_arch of full_adder1 is
16
17    component AND2
18    port(I0,I1: in STD_LOGIC;
19    O: out STD_LOGIC);
20    end component;
21
22    component OR3
23    port(I0,I1,I2: in STD_LOGIC;
24    O: out STD_LOGIC);
25    end component;
26
27    component XOR3
28    port(I0,I1,I2: in STD_LOGIC;
29    O: out STD_LOGIC);
30    end component;
31
32    signal C0,C1,C2: STD_LOGIC;
33
34 begin
35
36 AND_0: AND2 port map(Cin,A,C0);
37 AND_1: AND2 port map(Cin,B,C1);
38 AND_2: AND2 port map(A,B,C2);
39
40 OR_0: OR3 port map(C0,C1,C2,Cout);
41 XOR_0: XOR3 port map(Cin,A,B,SUM);
42
43 end adderF1_arch;
```

Mux 4_1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity mux4_1 is
8      Port ( D0 : in  STD_LOGIC;
9             D1 : in  STD_LOGIC;
10            D2 : in  STD_LOGIC;
11            D3 : in  STD_LOGIC;
12            S0 : in  STD_LOGIC;
13            S1 : in  STD_LOGIC;
14            Y : out  STD_LOGIC);
15 end mux4_1;
16
17 architecture mux_arch of mux4_1 is
18
19    component INV
20    port(I: in STD_LOGIC;
21    O: out STD_LOGIC);
22    end component;
23
24    component AND3
25    port(I0,I1,I2: in STD_LOGIC;
26    O: out STD_LOGIC);
27    end component;
28
29    component OR4
30    port(I0,I1,I2,I3: in STD_LOGIC;
31    O: out STD_LOGIC);
32    end component;
33
34    signal S0N,S1N,P0,P1,P2,P3: STD_LOGIC;
35 begin
36
37 INV_0: INV port map(S0, S0N);
38 INV_1: INV port map(S1, S1N);
39
40 AND_1: AND3 port map(S0N,S1N,D0,P0);
41 AND_2: AND3 port map(S0,S1N,D1,P1);
42 AND_3: AND3 port map(S0N,S1,D2,P2);
43 AND_4: AND3 port map(S0,S1,D3,P3);
44
45  OR_1: OR4 port map(P0,P1,P2,P3,Y);
46
47 end mux_arch;
```

Mux 4_1 Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY mux4_1_testbench IS
6  END mux4_1_testbench;
7
8  ARCHITECTURE behavior OF mux4_1_testbench IS
9      -- Component Declaration for the Unit Under Test (UUT)
10
11     COMPONENT mux4_1
12     PORT(
13          D0 : IN  std_logic;
14          D1 : IN  std_logic;
15          D2 : IN  std_logic;
16          D3 : IN  std_logic;
17          S0 : IN  std_logic;
18          S1 : IN  std_logic;
19          Y : OUT  std_logic
20         );
21     END COMPONENT;
23    --Inputs
24    signal D0 : std_logic := '0';
25    signal D1 : std_logic := '0';
26    signal D2 : std_logic := '0';
27    signal D3 : std_logic := '0';
28    signal S0 : std_logic := '0';
29    signal S1 : std_logic := '0';
30
31    --Outputs
32    signal Y : std_logic;
33    -- No clocks detected in port list. Replace <clock> below with
34    -- appropriate port name
35
36 BEGIN
37
38    -- Instantiate the Unit Under Test (UUT)
39    uut: mux4_1 PORT MAP (
40           D0 => D0,
41           D1 => D1,
42           D2 => D2,
43           D3 => D3,
44           S0 => S0,
```

```
          S1 => S1,
          Y => Y
        );

   -- Stimulus process
   stim_proc: process
   begin
   --Initialize inputs
      D0 <= '0'; D1 <= '0'; D2 <= '0'; D3 <= '0';
      --Select D0
      S1 <= '0'; S0 <= '0'; wait for 10 ns;
      D0 <= '1'; wait for 10 ns;
      D0 <= '0'; wait for 10 ns;
      --Select D1
      S1 <= '0'; S0 <= '1'; wait for 10 ns;
      D1 <= '1'; wait for 10 ns;
      D1 <= '0'; wait for 10 ns;
      --Select D2
      S1 <= '1'; S0 <= '0'; wait for 10 ns;
      D2 <= '1'; wait for 10 ns;
      D2 <= '0'; wait for 10 ns;
      --Select D3
       S1 <= '1'; S0 <= '1'; wait for 10 ns;
       D3 <= '1'; wait for 10 ns;
       D3 <= '0'; wait for 10 ns;
       wait;
    end process;

END;
```

Mux 5

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity MUX5 is
    Port ( I_Source_0 : in  STD_LOGIC_VECTOR (4 downto 0);
           I_Source_1 : in  STD_LOGIC_VECTOR (4 downto 0);
           I_Mux_Sel : in  STD_LOGIC;
           O_Mux : out  STD_LOGIC_VECTOR (4 downto 0));
end MUX5;

architecture MUX5_arch of MUX5 is

begin
   process(I_Source_0, I_Source_1, I_Mux_Sel)
      begin
         if(I_Mux_Sel = '0') then
            O_Mux <= I_Source_0;
         else
            O_Mux <= I_Source_1;
```

```
23          end if;
24      end process;
25
26  end MUX5_arch;
```

Mux5 Testbench

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY MUX5_testbench IS
6   END MUX5_testbench;
7
8   ARCHITECTURE behavior OF MUX5_testbench IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT MUX5
13      PORT(
14           I_Source_0 : IN  std_logic_vector(4 downto 0);
15           I_Source_1 : IN  std_logic_vector(4 downto 0);
16           I_Mux_Sel : IN  std_logic;
17           O_Mux : OUT  std_logic_vector(4 downto 0)
18          );
19      END COMPONENT;
20
21
22     --Inputs
23     signal I_Source_0 : std_logic_vector(4 downto 0) := (others => '0');
24     signal I_Source_1 : std_logic_vector(4 downto 0) := (others => '0');
25     signal I_Mux_Sel : std_logic := '0';
26     signal O_Mux : std_logic_vector(4 downto 0) := (others => '0');
27
28  BEGIN
29
30     -- Instantiate the Unit Under Test (UUT)
31     uut: MUX5 PORT MAP (
32            I_Source_0 => I_Source_0,
33            I_Source_1 => I_Source_1,
34            I_Mux_Sel => I_Mux_Sel,
35            O_Mux => O_Mux
36          );
37
38     -- Stimulus process
39     stim_proc: process
40     begin
41        I_Source_0 <= std_logic_vector(to_unsigned(300,5));
42        I_Source_1 <= std_logic_vector(to_unsigned(990,5));
43        I_Mux_Sel <= '1';
44        wait for 10 ns;
```

```
45        I_Mux_Sel <= '0';
46        wait;
47     end process;
48
49  END;
```

Mux32

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.NUMERIC_STD.ALL;
 4  library UNISIM;
 5  use UNISIM.VComponents.all;
 6
 7  entity MUX32 is
 8      Port ( I_Source_0 : in  STD_LOGIC_VECTOR (31 downto 0);
 9             I_Source_1 : in  STD_LOGIC_VECTOR (31 downto 0);
10             I_Mux_Sel : in  STD_LOGIC;
11             O_Mux : out  STD_LOGIC_VECTOR (31 downto 0));
12  end MUX32;
13
14  architecture MUX32_arch of MUX32 is
15
16  begin
17     process(I_Source_0, I_Source_1, I_Mux_Sel)
18
19     begin
20        if(I_Mux_Sel = '0') then
21           O_Mux <= I_Source_0;
22        else
23           O_Mux <= I_Source_1;
24        end if;
25     end process;
26
27  end MUX32_arch;
```

Mux32 Testbench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY MUX32_testbench IS
END MUX32_testbench;

ARCHITECTURE behavior OF MUX32_testbench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT MUX32
    PORT(
         I_Source_0 : IN  std_logic_vector(31 downto 0);
         I_Source_1 : IN  std_logic_vector(31 downto 0);
         I_Mux_Sel : IN  std_logic;
         O_Mux : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
```

ch

```
   signal I_Source_0 : std_logic_vector(31 downto 0) := (others => '0');
   signal I_Source_1 : std_logic_vector(31 downto 0) := (others => '0');
   signal I_Mux_Sel : std_logic := '0';

   --Outputs
   signal O_Mux : std_logic_vector(31 downto 0);

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: MUX32 PORT MAP (
          I_Source_0 => I_Source_0,
          I_Source_1 => I_Source_1,
          I_Mux_Sel => I_Mux_Sel,
          O_Mux => O_Mux
        );

   -- Stimulus process
   stim_proc: process
   begin
      I_Source_0 <= std_logic_vector(to_unsigned(333,32));
      I_Source_1 <= std_logic_vector(to_unsigned(993,32));
      I_Mux_Sel <= '0';
      wait for 10 ns;
      I_Mux_Sel <= '1';
      wait;
   end process;

END;
```

PC

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity PC is
8      Port ( I_PC : in  STD_LOGIC_VECTOR(31 downto 0);
9             I_PC_UPDATE : in  STD_LOGIC;
10            O_PC : out STD_LOGIC_VECTOR(31 downto 0));
11 end PC;
12
13 architecture PC_Arch of PC is
14
15 begin
16    process(I_PC_UPDATE)
17    begin
18       if I_PC_UPDATE = '1' then
19          if I_PC = "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU" then
20             O_PC <= "00000000000000000000000000000000";
21          else
22             O_PC <= I_PC;
23          end if;
24       end if;
25    end process;
26 end PC_Arch;
```

PC_INC

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity PC_INC is
8      Port ( I_PC : in  STD_LOGIC_VECTOR(31 downto 0);
9             O_PC : out  STD_LOGIC_VECTOR(31 downto 0));
10 end PC_INC;
11
12 architecture PC_INC_Arch of PC_INC is
13 begin
14
15    O_PC <= std_logic_vector(unsigned(I_PC) + 4);
16
17 end PC_INC_Arch;
```

RAM

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6  use STD.textio.all;
7
8  entity RAM is
9      Port ( I_RAM_EN : in  STD_LOGIC;
10            I_RAM_RE : in  STD_LOGIC;
11            I_RAM_WE : in  STD_LOGIC;
12            I_RAM_ADDR : in  STD_LOGIC_VECTOR (31 downto 0);
13            I_RAM_DATA : in  STD_LOGIC_VECTOR (31 downto 0);
14            O_RAM_DATA : out  STD_LOGIC_VECTOR (31 downto 0));
15 end RAM;
16
17 architecture RAM_Arch of RAM is
18    type buf64x32 is array (0 to 255) of STD_LOGIC_VECTOR(7 downto 0);
19    signal mem : buf64x32;
20
21    impure function init_ram(FileName : in string) return buf64x32 is
22       file fp: text;
23     variable ram_count : integer :=0;
24     variable temp_mem : buf64x32 := (others => x"00");
25     variable line_cache : line;
26     variable byte_cache : bit_vector (31 downto 0) :=x"00000000";
27  begin
28     file_open(fp, FileName, read_mode);
29     while not endfile(fp) loop
30        readline(fp, line_cache);
31        read(line_cache, byte_cache);
32        temp_mem(ram_count) := to_stdlogicvector(byte_cache(31 downto 24));
33        temp_mem(ram_count+1) := to_stdlogicvector(byte_cache(23 downto 16));
34        temp_mem(ram_count+2) := to_stdlogicvector(byte_cache(15 downto 8));
35        temp_mem(ram_count+3) := to_stdlogicvector(byte_cache(7 downto 0));
36        ram_count := ram_count+4;
37        if ram_count >= 255 then
38           exit;
39        end if;
40     end loop;
41     file_close(fp);
42     return temp_mem;
43  end function;
```

```
44
45  begin
46
47  process(I_RAM_EN, I_RAM_RE, I_RAM_WE, I_RAM_ADDR)
48     begin
49        mem <= init_ram("RAM_init.txt");
50        if I_RAM_EN = '1' then
51           if I_RAM_RE = '1' then
52              O_RAM_DATA(31 downto 24) <= mem(to_integer(unsigned(I_RAM_ADDR)) - 8192);
53              O_RAM_DATA(23 downto 16) <= mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+1);
54              O_RAM_DATA(15 downto 8) <= mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+2);
55              O_RAM_DATA(7 downto 0) <= mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+3);
56           end if;
57           if I_RAM_WE = '1' then
58              mem(to_integer(unsigned(I_RAM_ADDR)) - 8192) <= I_RAM_DATA(31 downto 24);
59              mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+1) <= I_RAM_DATA(23 downto 16);
60              mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+2) <= I_RAM_DATA(15 downto 8);
61              mem(to_integer(unsigned(I_RAM_ADDR)) - 8192+3) <= I_RAM_DATA(7 downto 0);
62           end if;
63        end if;
64     end process;
65  end RAM_Arch;
```

Ram Testbench

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY RAM_TestBench IS
6   END RAM_TestBench;
7
8   ARCHITECTURE behavior OF RAM_TestBench IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT RAM
13      PORT(
14           I_RAM_EN : IN  std_logic;
15           I_RAM_RE : IN  std_logic;
16           I_RAM_WE : IN  std_logic;
17           I_RAM_ADDR : IN  std_logic_vector(31 downto 0);
18           I_RAM_DATA : IN  std_logic_vector(31 downto 0);
19           O_RAM_DATA : OUT  std_logic_vector(31 downto 0)
20          );
21      END COMPONENT;
22
```

```
23
24      --Inputs
25      signal I_RAM_EN : std_logic := '0';
26      signal I_RAM_RE : std_logic := '0';
27      signal I_RAM_WE : std_logic := '0';
28      signal I_RAM_ADDR : std_logic_vector(31 downto 0) := (others => '0');
29      signal I_RAM_DATA : std_logic_vector(31 downto 0) := (others => '0');
30
31      --Outputs
32      signal O_RAM_DATA : std_logic_vector(31 downto 0);
33      -- No clocks detected in port list. Replace <clock> below with
34      -- appropriate port name
35
36
37  BEGIN
38
39      -- Instantiate the Unit Under Test (UUT)
40      uut: RAM PORT MAP (
41              I_RAM_EN => I_RAM_EN,
42              I_RAM_RE => I_RAM_RE,
43              I_RAM_WE => I_RAM_WE,
44              I_RAM_ADDR => I_RAM_ADDR,
45               I_RAM_DATA => I_RAM_DATA,
46               O_RAM_DATA => O_RAM_DATA
47            );
48
49
50  -- Stimulus process
51  stim_proc: process
52  variable num_lines : integer := 32;
53  -- the RAM range is 0x00002000 - 0x000020FF
54  -- that is 8192 to 8447 in decimal
55  variable address : integer := 8192;
56  begin
57      -- hold reset state for 10 ns.
58      wait for 10 ns;
59      -- Test #01
60      -- read the first 128 Bytes of the RAM
61      -- i.e. from 8192 to 8319
62      -- 0x00002000 - 0x0000207F
63      for i in 0 to (num_lines-1) loop
64          I_RAM_EN <= '1';
65          I_RAM_RE <= '1';
66          I_RAM_WE <= '0';
```

```
67          I_RAM_ADDR <= std_logic_vector(to_unsigned(address, I_RAM_ADDR'length));
68          wait for 10 ns;
69          I_RAM_EN <= '0';
70          address := address + 4;
71          wait for 10 ns;
72      end loop;
73      -- stop the RAM
74      I_RAM_EN <= '0';
75      I_RAM_RE <= '0';
76      I_RAM_WE <= '0';
77      wait for 10 ns;
78      -- TEST #02
79      -- write the first 8 bytes of the RAM without WE asserted
80      -- i.e. from 8192 to 8199
81      -- 0x00002000 - 0x00002007
82      I_RAM_EN <= '1';
83      I_RAM_RE <= '1';
84      I_RAM_WE <= '0';
85      I_RAM_ADDR <= x"00002000";
86      I_RAM_DATA <= x"66778899";
87      wait for 5 ns;
88      I_RAM_EN <= '0';
89      wait for 5 ns;
90      I_RAM_EN <= '1';
91      I_RAM_ADDR <= x"00002004";
92      I_RAM_DATA <= x"AABBCCDD";
93      wait for 5 ns;
94      I_RAM_EN <= '0';
95      wait for 5 ns;
96      -- stop the RAM
97      I_RAM_EN <= '0';
98      I_RAM_RE <= '0';
99      I_RAM_WE <= '0';
100     wait for 10 ns;
101     -- Test #03
102     -- read the first 8 bytes of the RAM
103     -- verify that the RAM content cannot be overwritten
104     -- if WE is not asserted
105     I_RAM_EN <= '1';
106     I_RAM_RE <= '1';
107     I_RAM_ADDR <= x"00002000";
108     wait for 10 ns;
109     I_RAM_EN <= '0';
110     wait for 10 ns;
```

```
111     I_RAM_EN <= '1';
112     I_RAM_ADDR <= x"00002004";
113     wait for 10 ns;
114     I_RAM_EN <= '0';
115     wait for 10 ns;
116     -- stop the RAM
117     I_RAM_EN <= '0';
118     I_RAM_RE <= '0';
119     I_RAM_WE <= '0';
120     wait for 10 ns;
121     -- Test #04
122     -- write the first 8 bytes of the RAM with WE asserted
123     -- i.e. from 8192 to 8199
124     -- 0x00002000 - 0x00002007
125     I_RAM_EN <= '1';
126     I_RAM_RE <= '0';
127     I_RAM_WE <= '1';
128     I_RAM_ADDR <= x"00002000";
129     I_RAM_DATA <= x"66778899";
130     wait for 5 ns;
131     I_RAM_EN <= '0';
132     wait for 5 ns;
133     I_RAM_EN <= '1';
134     I_RAM_RE <= '0';
135     I_RAM_WE <= '1';
136     I_RAM_ADDR <= x"00002004";
137     I_RAM_DATA <= x"AABBCCDD";
138     wait for 5 ns;
139     I_RAM_EN <= '0';
140     wait for 5 ns;
141     -- stop the RAM
142     I_RAM_EN <= '0';
143     I_RAM_RE <= '0';
144     I_RAM_WE <= '0';
145     wait for 10 ns;
146     -- Test #05
147     -- read the first 8 bytes of the RAM
148     -- verify that the RAM content can be updated when needed
149     I_RAM_EN <= '1';
150     I_RAM_RE <= '1';
151     I_RAM_WE <= '0';
152     I_RAM_ADDR <= x"00002000";
153     I_RAM_DATA <= x"00000000";
154     wait for 10 ns;
```

```
155     I_RAM_EN <= '0';
156     wait for 10 ns;
157     I_RAM_EN <= '1';
158     I_RAM_RE <= '1';
159     I_RAM_WE <= '0';
160     I_RAM_ADDR <= x"00002004";
161     wait for 10 ns;
162     -- end of the test
163     -- disable the module
164     I_RAM_EN <= '0';
165     I_RAM_RE <= '0';
166     I_RAM_RE <= '0';
167     wait;
168 end process;
169
170 END;
```

REG

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6  use STD.textio.all;
7
8  entity REG is
9      Port ( I_REG_EN : in  STD_LOGIC;
10            I_REG_WE : in  STD_LOGIC;
11            I_REG_SEL_RS : in  STD_LOGIC_VECTOR (4 downto 0);
12            I_REG_SEL_RT : in  STD_LOGIC_VECTOR (4 downto 0);
13            I_REG_SEL_RD : in  STD_LOGIC_VECTOR (4 downto 0);
14            I_REG_DATA_RD : in  STD_LOGIC_VECTOR (31 downto 0);
15            O_REG_DATA_A : out  STD_LOGIC_VECTOR (31 downto 0);
16            O_REG_DATA_B : out  STD_LOGIC_VECTOR (31 downto 0));
17 end REG;
18 --signal mem
19 architecture REG_Arch of REG is
20    type buf32x32 is array (0 to 31) of STD_LOGIC_VECTOR(31 downto 0);
21    signal mem : buf32x32;
22
```

```
23      impure function init_buf(FileName : in string) return buf32x32 is
24         constant LINE_NUM : integer :=32;
25         file fp: text;
26         variable temp_mem : buf32x32 := (others => x"00000000");
27         variable line_cache : line;
28         variable byte_cache : bit_vector (31 downto 0) :=x"00000000";
29      begin
30         file_open(fp, FileName, read_mode);
31         for i in 0 to LINE_NUM loop
32            if endfile(fp) then
33               exit;
34            else
35               readline(fp, line_cache);
36               read(line_cache, byte_cache);
37               temp_mem(i) := to_stdlogicvector(byte_cache);
38            end if;
39         end loop;
40         file_close(fp);
41         return temp_mem;
42      end function;
43
44   begin
45      process(I_REG_EN, I_REG_WE)
46      begin
47         mem <= init_buf("REG_init.txt");
48         if I_REG_EN = '1' then
49            if I_REG_WE = '1' then
50               if to_integer(unsigned(I_REG_SEL_RD)) /= 0 then
51                  mem(to_integer(unsigned(I_REG_SEL_RD))) <= I_REG_DATA_RD;
52               end if;
53            end if;
54            O_REG_DATA_A <= mem(to_integer(unsigned(I_REG_SEL_RS)));
55            O_REG_DATA_B <= mem(to_integer(unsigned(I_REG_SEL_RT)));
56         end if;
57      end process;
58   end REG_Arch;
```

REG Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY REG_TestBench IS
6  END REG_TestBench;
7
8  ARCHITECTURE behavior OF REG_TestBench IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT REG
13     PORT(
14          I_REG_EN : IN  std_logic;
15          I_REG_WE : IN  std_logic;
16          I_REG_SEL_RS : IN  std_logic_vector(4 downto 0);
17          I_REG_SEL_RT : IN  std_logic_vector(4 downto 0);
18          I_REG_SEL_RD : IN  std_logic_vector(4 downto 0);
19          I_REG_DATA_RD : IN  std_logic_vector(31 downto 0);
20          O_REG_DATA_A : OUT  std_logic_vector(31 downto 0);
21          O_REG_DATA_B : OUT  std_logic_vector(31 downto 0)
22         );
23      END COMPONENT;
24
25
26     --Inputs
27     signal I_REG_EN : std_logic := '0';
28     signal I_REG_WE : std_logic := '0';
29     signal I_REG_SEL_RS : std_logic_vector(4 downto 0) := (others => '0');
30     signal I_REG_SEL_RT : std_logic_vector(4 downto 0) := (others => '0');
31     signal I_REG_SEL_RD : std_logic_vector(4 downto 0) := (others => '0');
32     signal I_REG_DATA_RD : std_logic_vector(31 downto 0) := (others => '0');
33
34     --Outputs
35     signal O_REG_DATA_A : std_logic_vector(31 downto 0);
36     signal O_REG_DATA_B : std_logic_vector(31 downto 0);
37     -- No clocks detected in port list. Replace <clock> below with
38     -- appropriate port name
39
40
41 BEGIN
42
43     -- Instantiate the Unit Under Test (UUT)
44     uut: REG PORT MAP (
```

```
45            I_REG_EN => I_REG_EN,
46            I_REG_WE => I_REG_WE,
47            I_REG_SEL_RS => I_REG_SEL_RS,
48            I_REG_SEL_RT => I_REG_SEL_RT,
49            I_REG_SEL_RD => I_REG_SEL_RD,
50            I_REG_DATA_RD => I_REG_DATA_RD,
51            O_REG_DATA_A => O_REG_DATA_A,
52            O_REG_DATA_B => O_REG_DATA_B
53          );
54
55
56   -- Stimululus process
57  stim_proc: process
58  variable index: integer := 0;
59  variable NUM_OF_REGISTER: integer := 32;
60  begin
61     -- hold reset state for 10 ns.
62     wait for 10 ns;
63     -- show all register values
64     for i in 0 to (NUM_OF_REGISTER/2-1) loop
65        I_REG_EN <= '1';
66        I_REG_SEL_RS <= std_logic_vector(to_unsigned(index, I_REG_SEL_RS'length));
67        I_REG_SEL_RT <= std_logic_vector(to_unsigned((index+1), I_REG_SEL_RS'length));
68        wait for 10 ns;
69        I_REG_EN <= '0';
70        index := index + 2;
71        wait for 10 ns;
72     end loop;
73     -- attempt to update register values with WE disabled
74     I_REG_WE <= '0';
75     for i in 0 to (NUM_OF_REGISTER-1) loop
76        I_REG_EN <= '1';
77        I_REG_SEL_RD <= std_logic_vector(to_unsigned(i, I_REG_SEL_RS'length));
78        I_REG_DATA_RD <= x"00000000";
79        wait for 5 ns;
80        I_REG_EN <= '0';
81        wait for 5 ns;
82     end loop;
83     -- show all register values
84     index := 0;
85     I_REG_WE <= '0';
86     for i in 0 to (NUM_OF_REGISTER/2-1) loop
87        I_REG_EN <= '1';
88        I_REG_SEL_RS <= std_logic_vector(to_unsigned(index, I_REG_SEL_RS'length));
```

```
89          I_REG_SEL_RT <= std_logic_vector(to_unsigned((index+1), I_REG_SEL_RS'length));
90          wait for 10 ns;
91          I_REG_EN <= '0';
92          index := index + 2;
93          wait for 10 ns;
94       end loop;
95       -- attempt to update register value with WE enabled
96       I_REG_WE <= '1';
97       for i in 0 to (NUM_OF_REGISTER-1) loop
98          I_REG_EN <= '1';
99          I_REG_SEL_RD <= std_logic_vector(to_unsigned(i, I_REG_SEL_RS'length));
100         I_REG_DATA_RD <= x"00000000";
101         wait for 5 ns;
102         I_REG_EN <= '0';
103         wait for 5 ns;
104      end loop;
105      -- show all register values
106      index := 0;
107      I_REG_WE <= '0';
108      for i in 0 to (NUM_OF_REGISTER/2-1) loop
109         I_REG_EN <= '1';
110         I_REG_SEL_RS <= std_logic_vector(to_unsigned(index, I_REG_SEL_RS'length));
111         I_REG_SEL_RT <= std_logic_vector(to_unsigned((index+1), I_REG_SEL_RS'length));
112         wait for 10 ns;
113         I_REG_EN <= '0';
114         index := index + 2;
115         wait for 10 ns;
116      end loop;
117      wait;
118   end process;
119
120   END;
```

ROM 1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6  use STD.textio.all;
7
8  entity ROM is
9      Port ( I_ROM_EN : in  STD_LOGIC;
10            I_ROM_ADDR : in  STD_LOGIC_VECTOR (31 downto 0);
11            O_ROM_DATA : out  STD_LOGIC_VECTOR (31 downto 0));
12 end ROM;
13
14 architecture ROM_Arch of ROM is
15    type buf64x32 is array (0 to 255) of STD_LOGIC_VECTOR(7 downto 0);
16
17    -- function to initialize ROM
18    impure function init_rom(FileName : in string) return buf64x32 is
19       file fp: text;
20       variable rom_count : integer :=0;
21       variable temp_mem : buf64x32 := (others => x"00");
22       variable line_cache : line;
```

```
23          variable byte_cache : bit_vector (31 downto 0) :=x"00000000";
24       begin
25          file_open(fp, FileName, read_mode);
26          while not endfile(fp) loop
27             readline(fp, line_cache);
28             read(line_cache, byte_cache);
29             temp_mem(rom_count) := to_stdlogicvector(byte_cache(31 downto 24));
30             temp_mem(rom_count+1) := to_stdlogicvector(byte_cache(23 downto 16));
31             temp_mem(rom_count+2) := to_stdlogicvector(byte_cache(15 downto 8));
32             temp_mem(rom_count+3) := to_stdlogicvector(byte_cache(7 downto 0));
33             rom_count := rom_count+4;
34             if rom_count >= 255 then
35                exit;
36             end if;
37          end loop;
38          file_close(fp);
39          return temp_mem;
40       end function;
41
42    --Initializing ROM memory
43    signal mem : buf64x32:= init_rom("ROM_init.txt");
44
45    begin
46       process(I_ROM_EN, I_ROM_ADDR)
47       begin
48       if I_ROM_EN = '1' then
49          O_ROM_DATA(31 downto 24) <= mem(to_integer(unsigned(I_ROM_ADDR)));
50          O_ROM_DATA(23 downto 16) <= mem(to_integer(unsigned(I_ROM_ADDR))+1);
51          O_ROM_DATA(15 downto 8) <= mem(to_integer(unsigned(I_ROM_ADDR))+2);
52          O_ROM_DATA(7 downto 0) <= mem(to_integer(unsigned(I_ROM_ADDR))+3);
53       end if;
54    end process;
55    end ROM_Arch;
```

ROM Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY ROM_TestBench IS
6  END ROM_TestBench;
7
8  ARCHITECTURE behavior OF ROM_TestBench IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT ROM
13     PORT(
14          I_ROM_EN : IN  std_logic;
15          I_ROM_ADDR : IN  std_logic_vector(31 downto 0);
16          O_ROM_DATA : OUT  std_logic_vector(31 downto 0)
17         );
18     END COMPONENT;
19
20
21    --Inputs
22    signal I_ROM_EN : std_logic := '0';
```

```
23     signal I_ROM_ADDR : std_logic_vector(31 downto 0) := (others => '0');
24
25     --Outputs
26     signal O_ROM_DATA : std_logic_vector(31 downto 0);
27     -- No clocks detected in port list. Replace <clock> below with
28     -- appropriate port name
29
30
31  BEGIN
32
33     -- Instantiate the Unit Under Test (UUT)
34     uut: ROM PORT MAP (
35            I_ROM_EN => I_ROM_EN,
36            I_ROM_ADDR => I_ROM_ADDR,
37            O_ROM_DATA => O_ROM_DATA
38          );
39
40
41  -- Stimulus process
42  stim_proc: process
43  variable num_lines : integer := 12;
44  variable address : integer := 0;
45  begin
46     -- hold reset state for 10 ns.
47     wait for 10 ns;
48     -- insert stimulus here
49     for i in 0 to num_lines loop
50        I_ROM_EN <= '1';
51        I_ROM_ADDR <= std_logic_vector(to_unsigned(address, I_ROM_ADDR'length));
52        wait for 10 ns;
53        I_ROM_EN <= '0';
54        address := address + 4;
55        wait for 10 ns;
56     end loop;
57     wait;
58  end process;
59
60  END;
```

Shift Left 2 (SL2)

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity SL2 is
generic(
DEPTH : integer := 32;
Shift_Magnitude: integer := 2
);
port(
   SI: in  std_logic_vector(31 downto 0);
   SO: out std_logic_vector(31 downto 0)
);
end SL2;

architecture SL2_Arch of SL2 is
signal shreg : std_logic_vector(DEPTH - 1 downto 0);
begin
   process(SI)
   begin
      shreg<=(SI(DEPTH-1-Shift_Magnitude downto 0) & "00");
   end process;
   SO <= shreg;
end SL2_Arch;
```

Sign Extension

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity SIGN_EXT is
    Port ( I_EXT_16 : in  STD_LOGIC_VECTOR (15 downto 0);
           O_EXT_32 : out  STD_LOGIC_VECTOR (31 downto 0));
end SIGN_EXT;

architecture SIGN_EXT_ARCH of SIGN_EXT is
begin
   process(I_EXT_16)
      begin
      if (I_EXT_16(15) = '1') then
         O_EXT_32 <= "1111111111111111" & I_EXT_16;
      else
         O_EXT_32 <= "0000000000000000" & I_EXT_16;
      end if;
   end process;
end SIGN_EXT_ARCH;
```

Sign Extension Test bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY SIGN_EXT_testbench IS
END SIGN_EXT_testbench;

ARCHITECTURE behavior OF SIGN_EXT_testbench IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT SIGN_EXT
    PORT(
         I_EXT_16 : IN  std_logic_vector(15 downto 0);
         O_EXT_32 : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal I_EXT_16 : std_logic_vector(15 downto 0) := (others => '0');

   --Outputs
   signal O_EXT_32 : std_logic_vector(31 downto 0);

BEGIN

   -- Instantiate the Unit Under Test (UUT)
   uut: SIGN_EXT PORT MAP (
          I_EXT_16 => I_EXT_16,
          O_EXT_32 => O_EXT_32
        );

   -- Stimulus process
   stim_proc: process
   begin
      I_EXT_16 <= "0000100000000000";
      wait for 10 ns;
      I_EXT_16 <= "1000000000000000";
      wait;
   end process;

END;

```

SL2 Testbench

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  USE ieee.numeric_std.ALL;
4
5  ENTITY SL2_Testbench IS
6  END SL2_Testbench;
7
8  ARCHITECTURE behavior OF SL2_Testbench IS
9
10     -- Component Declaration for the Unit Under Test (UUT)
11
12     COMPONENT SL2
13     PORT(
14          SI : IN  std_logic_vector(31 downto 0);
15          SO : OUT  std_logic_vector(31 downto 0)
16         );
17     END COMPONENT;
18
19
20    --Inputs
21    signal SI : std_logic_vector(31 downto 0) := (others => '0');
22
23    --Outputs
24    signal SO : std_logic_vector(31 downto 0);
25    -- No clocks detected in port list. Replace <clock> below with
26    -- appropriate port name
27
28 BEGIN
29
30    -- Instantiate the Unit Under Test (UUT)
31    uut: SL2 PORT MAP (
32           SI => SI,
33           SO => SO
34         );
35
36
37    -- Stimulus process
38    stim_proc: process
39    begin
40    SI <= "00000000000000000000000000000010";
41
42       -- insert stimulus here
43
44       wait;
```

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY SL2_Testbench IS
END SL2_Testbench;

ARCHITECTURE behavior OF SL2_Testbench IS
    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT SL2
    PORT(
         SI : IN  std_logic_vector(31 downto 0);
         SO : OUT  std_logic_vector(31 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal SI : std_logic_vector(31 downto 0) := (others => '0');

   --Outputs
    signal SO : std_logic_vector(31 downto 0);

BEGIN

    -- Instantiate the Unit Under Test (UUT)
    uut: SL2 PORT MAP (
            SI => SI,
            SO => SO
          );


    -- Stimulus process
    stim_proc: process
    begin
    SI <= "00000000000000000000000000000010";


       wait;
    end process;
END;
```

Test Module

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4  library UNISIM;
5  use UNISIM.VComponents.all;
6
7  entity Test_Module is
8      Port ( I_EN : in  STD_LOGIC;
9             I_Instr : in  STD_LOGIC_VECTOR (31 downto 0);
10            O_RegDst : out  STD_LOGIC;
11            O_Jump : out  STD_LOGIC;
12            O_Beq : out  STD_LOGIC;
13            O_Bne : out  STD_LOGIC;
14            O_MemRead : out  STD_LOGIC;
15            O_MemtoReg : out  STD_LOGIC;
16            O_MemWrite : out  STD_LOGIC;
17            O_ALUSrc : out  STD_LOGIC;
18            O_RegWrite : out  STD_LOGIC;
19            O_ALUCtl : out  STD_LOGIC_VECTOR (3 downto 0)
20          );
21 end Test_Module;
22
23 architecture Test_Module_arch of Test_Module is
24     COMPONENT Dec PORT ( I_DEC_EN : in STD_LOGIC;
25                          I_DEC_OpCode : in STD_LOGIC_VECTOR (5 downto 0);
26                          O_DEC_RegDst : out STD_LOGIC;
27                          O_DEC_Jump : out STD_LOGIC;
28                          O_DEC_Beq : out STD_LOGIC;
29                          O_DEC_Bne : out STD_LOGIC;
30                          O_DEC_MemRead : out STD_LOGIC;
31                          O_DEC_MemtoReg : out STD_LOGIC;
32                          O_DEC_ALUOp : out STD_LOGIC_VECTOR (1 downto 0);
33                          O_DEC_MemWrite : out STD_LOGIC;
34                          O_DEC_ALUSrc : out STD_LOGIC;
35                          O_DEC_RegWrite : out STD_LOGIC
36                        );
37     END COMPONENT;
38
39     COMPONENT Acu PORT ( I_ACU_ALUOp : in STD_LOGIC_VECTOR (1 downto 0);
40                          I_ACU_Funct : in  STD_LOGIC_VECTOR (5 downto 0);
41                          O_ACU_CTL : out STD_LOGIC_VECTOR (3 downto 0)
42                        );
43     END COMPONENT;
44
```

```
45  signal ALUOp: STD_LOGIC_VECTOR (1 downto 0);
46  begin
47
48     DEC1: Dec PORT MAP ( I_EN,
49                          I_Instr (31 downto 26),
50                          O_RegDst,
51                          O_Jump,
52                          O_Beq,
53                          O_Bne,
54                          O_MemRead,
55                          O_MemtoReg,
56                          ALUOp,
57                          O_MemWrite,
58                          O_ALUSrc,
59                          O_RegWrite
60                        );
61
62     ACU1: Acu PORT MAP ( ALUOp,
63                          I_Instr (5 downto 0),
64                          O_ALUCtl
65                        );
66  end Test_Module_arch;
```

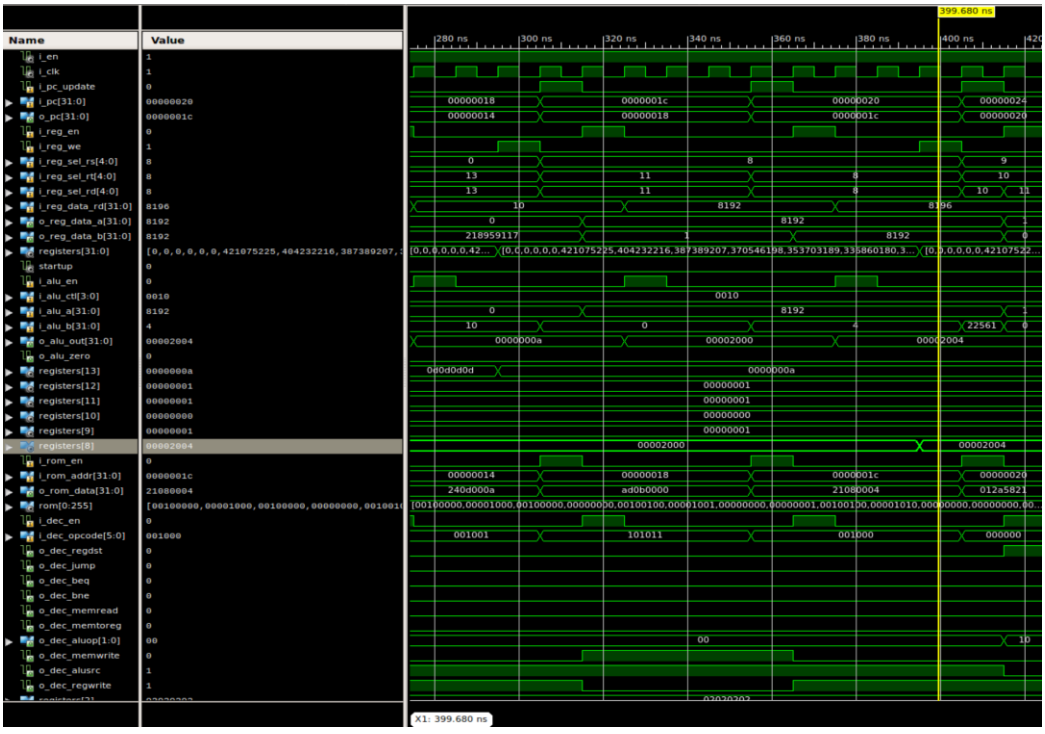
Test_Module Testbench

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   USE ieee.numeric_std.ALL;
4
5   ENTITY Test_Module_testbench IS
6   END Test_Module_testbench;
7
8   ARCHITECTURE behavior OF Test_Module_testbench IS
9
10      -- Component Declaration for the Unit Under Test (UUT)
11
12      COMPONENT Test_Module
13      PORT(
14           I_EN : IN  std_logic;
15           I_Instr : IN  std_logic_vector(31 downto 0);
16           O_RegDst : OUT  std_logic;
17           O_Jump : OUT  std_logic;
18           O_Beq : OUT  std_logic;
19           O_Bne : OUT  std_logic;
20           O_MemRead : OUT  std_logic;
21           O_MemtoReg : OUT  std_logic;
22           O_MemWrite : OUT  std_logic;
```

```
23          O_ALUSrc : OUT  std_logic;
24          O_RegWrite : OUT  std_logic;
25          O_ALUCtl : OUT  std_logic_vector(3 downto 0)
26        );
27     END COMPONENT;
28
29
30    --Inputs
31    signal I_EN : std_logic := '0';
32    signal I_Instr : std_logic_vector(31 downto 0) := (others => '0');
33
34    --Outputs
35    signal O_RegDst : std_logic;
36    signal O_Jump : std_logic;
37    signal O_Beq : std_logic;
38    signal O_Bne : std_logic;
39    signal O_MemRead : std_logic;
40    signal O_MemtoReg : std_logic;
41    signal O_MemWrite : std_logic;
42    signal O_ALUSrc : std_logic;
43    signal O_RegWrite : std_logic;
44    signal O_ALUCtl : std_logic_vector(3 downto 0);
45
46
47 BEGIN
48
49    -- Instantiate the Unit Under Test (UUT)
50    uut: Test_Module PORT MAP (
51           I_EN => I_EN,
52           I_Instr => I_Instr,
53           O_RegDst => O_RegDst,
54           O_Jump => O_Jump,
55           O_Beq => O_Beq,
56           O_Bne => O_Bne,
57           O_MemRead => O_MemRead,
58           O_MemtoReg => O_MemtoReg,
59           O_MemWrite => O_MemWrite,
60           O_ALUSrc => O_ALUSrc,
61           O_RegWrite => O_RegWrite,
62           O_ALUCtl => O_ALUCtl
63         );
64
65    -- Stimulus process
66    stim_proc: process
```

```
67      begin
68          I_EN <= '0';
69          wait for 20 ns;
70          I_EN <= '1';
71          I_Instr <= X"012a5821";
72          wait for 20 ns;
73          I_Instr <= X"24090001";
74          wait for 20 ns;
75          I_Instr <= X"12120004";
76          wait for 20 ns;
77          I_Instr <= X"8e080000";
78          wait for 20 ns;
79          I_Instr <= X"15acfff9";
80          wait for 20 ns;
81          I_Instr <= X"ae08fffc";
82          wait for 20 ns;
83          I_Instr <= X"08100005";
84          wait;
85      end process;
86
87  END;
```

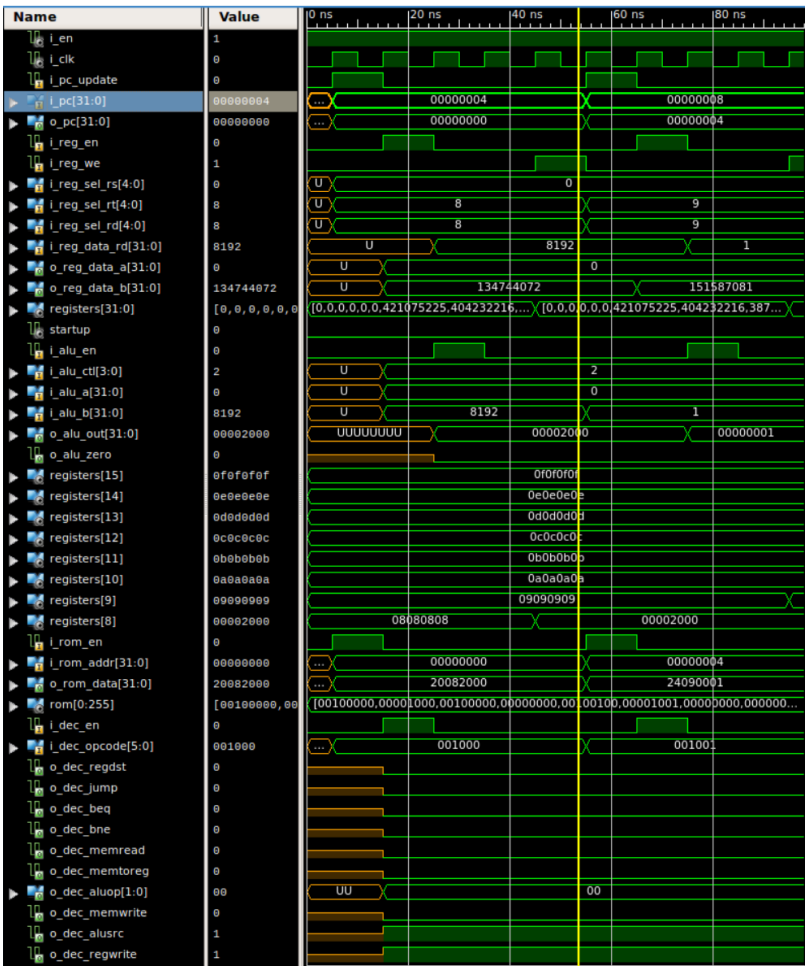
INSTRUCTION EXPLANATIONS



Instruction code: 00100001000010000000000000000100

The first instruction our processor handles is the addi instruction (OPCODE = 001000). This instruction will add an immediate value to the value in register $t0. Bits 15-0 of the instruction are sign extended and added to the selected register. In this scenario, the immediate value is 0x00000004, and the value in $t0 is 0x00002000. Also, the selected register is also $t0, so the result will be saved back to $t0 in register file 8.

Signal	Value	Note
regdst	0	Not written to a register
bne	0	Does not use bne
bqe	0	Does not use beq
memread	0	Not read from memory
memtoreg	0	Not loading from memory and writing to a register
aluop	0b00	Load word
memwrite	0	Not writing to memory
alusrc	1	Sign extended immediate
regwrite	1	Writing to a register
Alu_ctr	0b0010	Addition operation on ALU
Reg_file[8]	0x00002000	Value in register $t0
Reg_file[8]	0x00002004	Updated value in $t0
Alu_out	0x00002004	Result of ALU addition
O_pc	0x0000001c	Current instruction address
I_pc	0x00000020	Next instruction address

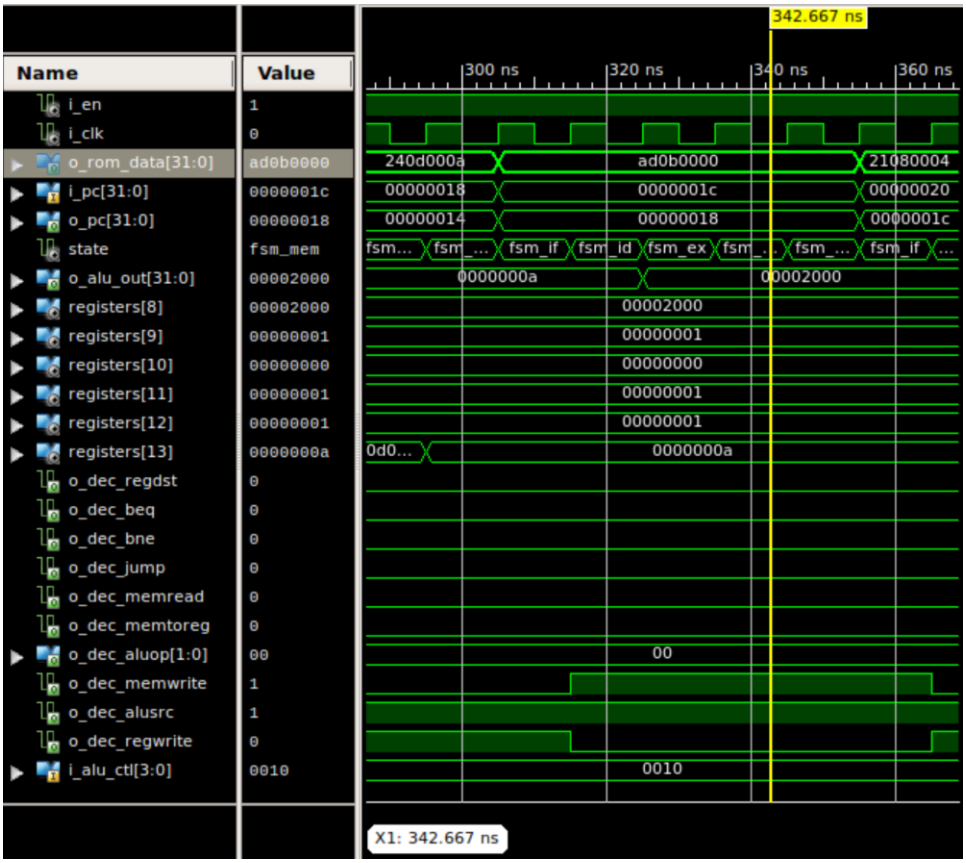


Instruction Code: 00100000000010000010000000000000

Our processor also supports the addiu instruction (OPCODE = 001001). This instruction is the same as addi in every sense except that the addends are treated as unsigned values. In this scenario, addi is being used in the psuedo instruction load word. The memory address of the immediate value is added into the selected register $t0.

Signal	Value	Note
regdst	0	Not written to a register
bne	0	Not using bne
bqe	0	Not using beq
memread	0	Not reading from memory
memtoreg	0	Not loading from memory and writing to a register
aluop	0b00	Load word
memwrite	0	Not written to memory
alusrc	1	Sign extended immediate
regwrite	1	Writing to a register

Alu_ctr	0b0010	Addition on ALU
Reg_file[8]	0x00002000	Value in $t0
Alu_out	0x00002000	Result of ALU addition
O_pc	0x00000000	Current instruction address
I_pc	0x00000004	Next instruction address

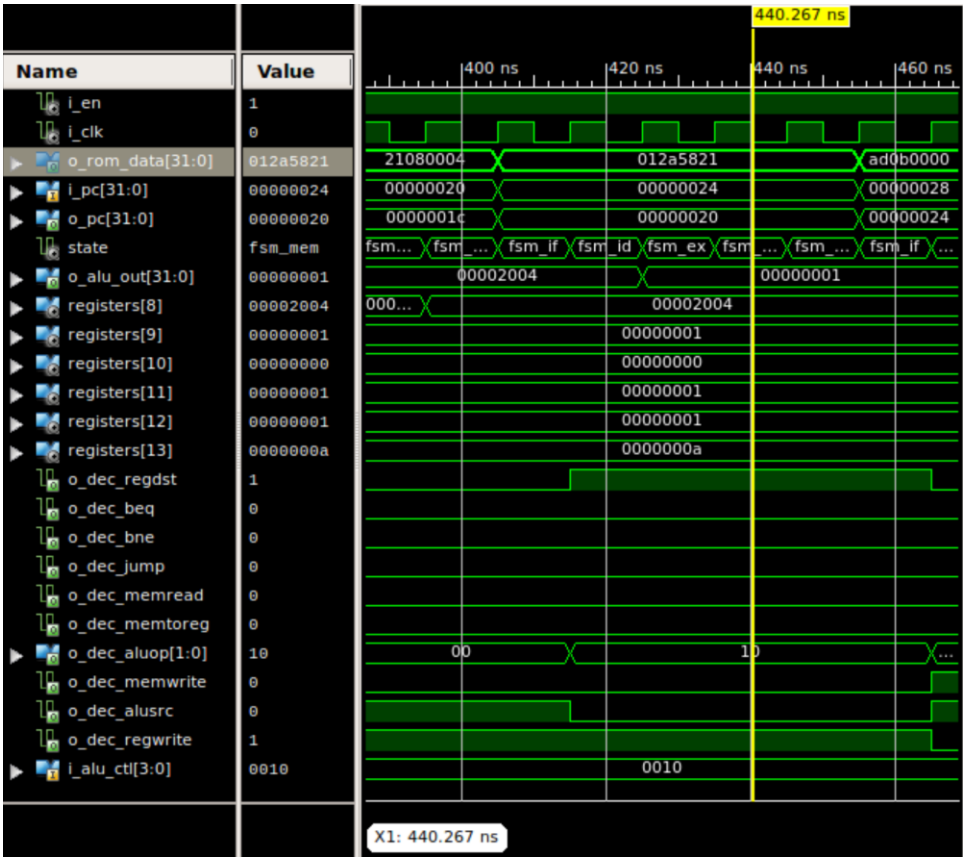


Instruction Code: 10101101000010110000000000000000

Our processor also supports the sw instruction (OPCODE = 101011). This instruction will save the data in a selected register at a given memory address. In this scenario, the value from 0($t0) is saved into register $t3.

Signal	Value	Note
regdst	0	Not written to a register
bne	0	Not using bne
bqe	0	Not using beq
memread	0	Not reading from memory
memtoreg	0	Not loading from memory and writing to a register
aluop	0b00	Load word
memwrite	1	written to memory
alusrc	1	Sign extended immediate

regwrite	0	Not writing to a register
Alu_ctr	0b0010	Addition on ALU
Reg_file[11]	0x00000001	Value in $t3
Reg_file[8]	0x00002000	Value in 0($t0)
Alu_out	0x00002000	Result of ALU addition
O_pc	0x00000018	Current instruction address
I_pc	0x0000001c	Next instruction address

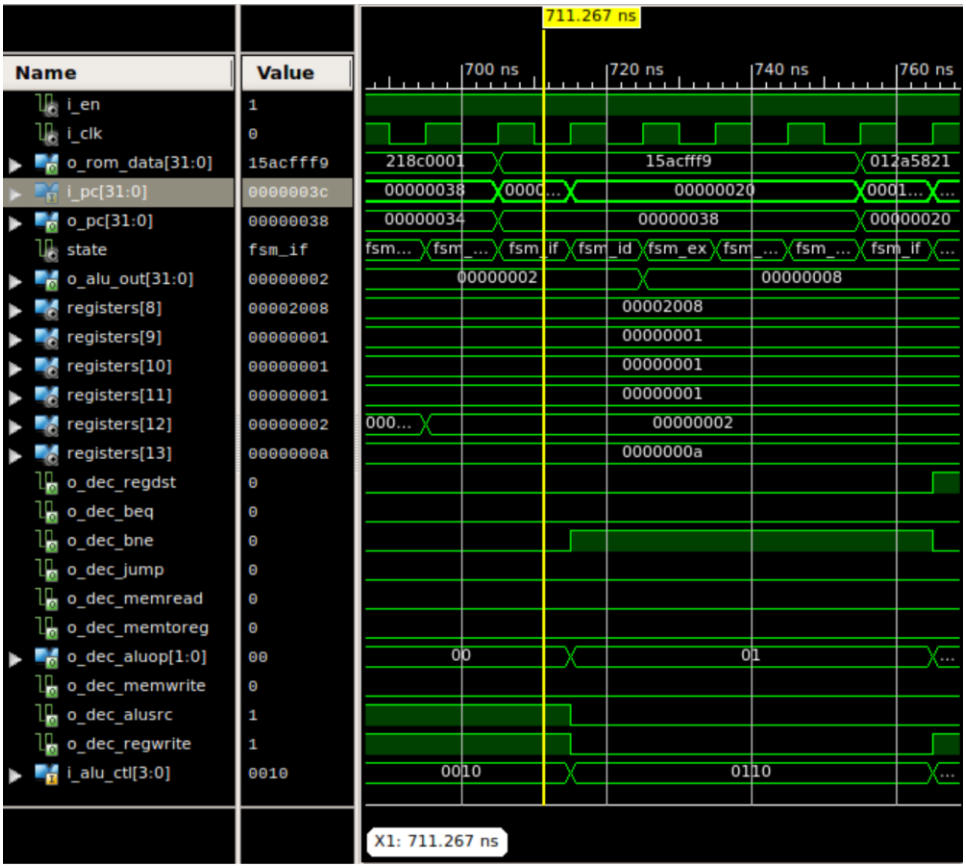


Instruction Code: 00000001001010100101100000100001

Our processor supports the addu instruction (OPCODE = 000000). This instruction adds two registers together while treating both values as unsigned, and then stores it into another register. In this case, the values in registers $t1 and $t2 are added together and stored into register $t3.

Signal	Value	Note
regdst	1	Writing to a register
bne	0	Not using bne
bqe	0	Not using beq
memread	0	Not reading from memory
memtoreg	0	Not loading from memory and writing to a register

aluop	0b10	R-type
memwrite	0	Not written to memory
alusrc	0	Second input of ALU is from the register
regwrite	1	Writing to a register
Alu_ctr	0b0010	Addition on ALU
Reg_file[11]	0x00000001	Contents of $t3
Reg_file[9]	0x00000001	Contents of $t1
Reg_file[10]	0x00000000	Contents of $t2
Alu_out	0x00000001	Result of ALU addition
O_pc	0x00000020	Current instruction address
I_pc	0x00000024	Next instruction address

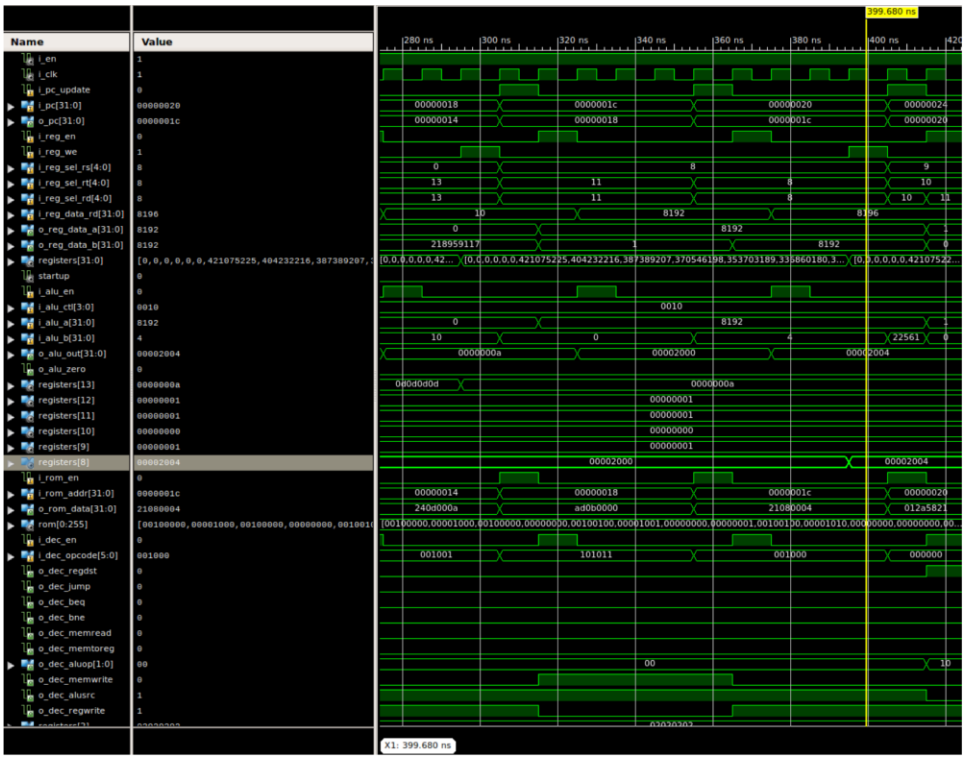


Instruction Code: 00010101101011001111111111111001

Our processor supports the bne instruction (OPCODE = 000101). This instruction will compare the values in two registers, and branch to a designated address should they not be equal. In this

case, the values within registers $t4 and $t5 are compared and are not equal. Therefore, we update the pc to the target label address.

Signal	Value	Note
regdst	0	Writing to a register
bne	1	using bne
bqe	0	Not using beq
memread	0	Not reading from memory
memtoreg	0	Not loading from memory and writing to a register
aluop	0b01	beq
memwrite	0	Not written to memory
alusrc	0	Second input of ALU is from the register
regwrite	0	Not Writing to a register
Alu_ctr	0b0110	subtraction on ALU
Reg_file[13]	0x0000000a	Contents of $t5
Reg_file[12]	0x00000002	Contents of $t4
Alu_out	0x00000008	Result of ALU addition
O_pc	0x00000038	Current instruction address
I_pc	0x0000003c	Next instruction address, adding 4
I_pc*	0x00000020	Correct address since its branch



Instruction Code: 00000000000000000000000000001100

Lastly, our processor supports the syscall instruction to terminate a program.

Signal	Value	Note
regdst	1	Not a R-type instruction
bne	0	Not using bne
bqe	0	Not using beq
memread	0	Not reading from memory
memtoreg	0	Not loading from memory and writing to a register
aluop	0b10	Addition is done on ALU but result won't be used due to program termination
memwrite	0	Not written to memory
alusrc	0	Second input of ALU is from the register
regwrite	1	Writing to a register, but not used
Alu_ctr	0b0010	Addition on ALU
Reg_file[2]	0x0000000a	Register 2 value
Alu_out	0x0000000a	Result of ALU addition
O_pc	0x00000040	Current instruction address
I_pc	0x00000044	Next instruction address

Data memory content with Fibonacci number

From the data memory content shown below when the processor finishes the MIPS program (registers shown on the left, and RAM shown on the right), the Fibonacci numbers are generated correctly. From starting at Fibonacci base number of 1, the 10th iteration of the RAM achieves the result of fib (10) = 55. We can also see with the register files that the register containing F(n) is 55 which is the correct value at n = 10. The RAM shows the result of each Fibonacci from 1 to 10.

Register 8 = $t0 = The register of the address of fib(n)
Register 9 = $t1 = The register holds fib(10)
Register 10 = $t2 = The register of fib(9) (This is after the values are copied over, so at the start of this loop it contained fib(8))
Register 11 = $t3 = The register of fib(10)
Register 12 = $t4 = The counter variable
Register 13 = $t5 = The wanted fib index.

13	10
12	10
11	55
10	34
9	55
8	8232

	0	1	2	3
0	0	0	0	1
4	0	0	0	1
8	0	0	0	2
12	0	0	0	3
16	0	0	0	5
20	0	0	0	8
24	0	0	0	13
28	0	0	0	21
32	0	0	0	34
36	0	0	0	55
40	0	0	0	0