32-Bit Microprocessor VHDL Codes

Names: Arif Emre YILDIZ, Samet YILMAZ

Top Design:

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

entity MIPS is
   port(
   clk : in std_logic;
   Reset: in std_logic;
   Mips_out: out std_logic_vector(31 downto 0);
   SegOut: out std_logic_vector(6 downto 0);
   Anodes: out std_logic_vector(3 downto 0)
   );
end MIPS;

architecture Behavioral of MIPS is

component ALU32Bit is
  Port (
-- INPUTS:
        A: in std_logic_vector (31 downto 0); --32-Bit input port A.
        B: in std_logic_vector (31 downto 0); --32-Bit input port B.
        ALUControl: in std_logic_vector (3 downto 0); --4-Bit input control bits to select an ALU operation.
-- OUTPUTS:
        ALUResult: out std_logic_vector (31 downto 0);  --32-Bit ALU result output.
        ZERO: out std_logic --1-Bit output flag.
);
end component;
```

```
component Controller is
PORT (
Op: in STD_LOGIC_VECTOR(5 DOWNTO 0);--The opcode from the current instruction
Funct: in STD_LOGIC_VECTOR(5 DOWNTO 0);
--which select specific data paths depending on the instruction
MemRead: out std_logic;
MemtoReg: out STD_LOGIC;
MemWrite: out STD_LOGIC;
ALUControl: out STD_LOGIC_VECTOR(3 DOWNTO 0);
ALUSrc: out STD_LOGIC;
RegDst: out STD_LOGIC;
RegWrite: out STD_LOGIC;
PCSrc: out std_logic
);
end component;

component dataMemory is
  port(
    clk  : in std_logic;
    memWrite  : in std_logic;
    memRead   : in std_logic;
    address      : in std_logic_vector(31 downto 0);
    writeData     : in std_logic_vector(31 downto 0);
    readData      : out std_logic_vector(31 downto 0)
  );
end component;

component instructionmemory is
---------//ports//--------------------------------------
PORT (address: in STD_LOGIC_VECTOR(31 DOWNTO 0);
```

```
            instruction: out STD_LOGIC_VECTOR(31 DOWNTO 0));
-----------------------------------------------
end component;

component ProgramCounter is
PORT(
Address: IN STD_LOGIC_VECTOR(31 downto 0); -- input
Reset : IN STD_LOGIC; -- async. clear.
Clk : IN STD_LOGIC; -- clock.
PCResult: OUT STD_LOGIC_VECTOR(31 downto 0) -- output.
);
end component;

component RegisterFile is
port(
   ReadRegister1: in std_logic_vector(4 downto 0);
   ReadRegister2: in std_logic_vector(4 downto 0);
   WriteRegister: in std_logic_vector(4 downto 0);
   WriteData: in std_logic_vector(31 downto 0);
   RegWrite: in std_logic;
   clk: in std_logic;

   ReadData1: out std_logic_vector(31 downto 0):=x"00000000";
   ReadData2: out std_logic_vector(31 downto 0):=x"00000000"
);
end component;

component SignExtension is

PORT (
A: in STD_LOGIC_VECTOR(15 DOWNTO 0);
```

```
SignImm: out STD_LOGIC_VECTOR(31 DOWNTO 0)
);

end component;

component PCAdder is
port(
   PCResult: in std_logic_vector (31 downto 0);
   PCAddResult: out std_logic_vector(31 downto 0)
);
end component;

component Mux is
   Port ( S : in STD_LOGIC;
          A : in STD_LOGIC_VECTOR (31 downto 0);
          B : in STD_LOGIC_VECTOR (31 downto 0);
          Y : out STD_LOGIC_VECTOR (31 downto 0));
end component;

component Mux5bit is
   Port ( SS : in STD_LOGIC;
          AA : in STD_LOGIC_VECTOR (4 downto 0);
          BB : in STD_LOGIC_VECTOR (4 downto 0);
          YY : out STD_LOGIC_VECTOR (4 downto 0));
end component;

component Adder is
   port(
       X, M : in std_logic_vector(31 downto 0);
       SUM : out std_logic_vector(31 downto 0)
   );
```

```
end component;

component shiftleft2 is
        port (
                entry: in std_logic_vector(31 downto 0);
                exitt: out std_logic_vector(31 downto 0)
        );
end component;

component ClkDiv is
   port ( clk: in std_logic;
      clkParameter: in integer;
      clock_out: out std_logic
   );
end component;

component FourDigit is
Port (
BCDin: in std_logic_vector (13 downto 0);
clk: in std_logic;
seven_segment : out std_logic_vector (6 downto 0);
anode: out std_logic_vector (3 downto 0)
);
end component;

component seven_MUX is
   Port ( S : in std_logic_vector(3 downto 0);
        B : in STD_LOGIC_VECTOR (31 downto 0);
        A: in std_logic_vector (31 downto 0);
        Y : out STD_LOGIC_VECTOR (31 downto 0):= x"00000000"
        );
```

```
end component;

------------------//signals//---------------------------------

-- signal a,b,c,d,e: std_logic; buraya signal yazcaz

signal rd_m4, alu_addm4, m4_wd, rd2_wdm2, rd1_aluA, m2_aluB, se_m4sl2, sl2_adder, adder_m3, m3_pc, pcadd_adderm3, pc_pcaddim: std_logic_vector(31 downto 0);

signal im_conreg, alugiris: std_logic_vector(31 downto 0);

signal m1_reg: std_logic_vector(4 downto 0);

signal con_mr, con_mtr, con_mw, con_rd, con_alusrc, con_rw, con_pcsrc, ZERO_led, systemCLK, displayCLK, clkIn: std_logic;

signal  con_alu, anodeOut: std_logic_vector(3 downto 0);

signal sevenOut,m5_Out: std_logic_vector(6 downto 0);



begin

clkIn <= clk;

Mips_out <= m4_wd;


ALU: ALU32Bit port map(

   A => alugiris,

         B => m2_aluB,

         ALUControl => con_alu,

         ALUResult => alu_addm4,

         ZERO => ZERO_led

);


Cont: Controller port map(

Op => im_conreg(31 downto 26),

Funct => im_conreg(5 downto 0),

MemRead => con_mr,

MemtoReg => con_mtr,

MemWrite => con_mw,

ALUControl => con_alu,
```

```
ALUSrc => con_alusrc,
RegDst => con_rd,
RegWrite => con_rw,
PCSrc => con_pcsrc
);

Data: dataMemory port map(
   clk => systemCLK,
   memWrite => con_mw,
   memRead => con_mr,
   address =>  alu_addm4,
   writeData =>  rd2_wdm2,
   readData => rd_m4
);

Inst: instructionmemory port map(
   address => pc_pcaddim,
          instruction => im_conreg
);

PC: ProgramCounter port map(
   Address => m3_pc,
   Reset => Reset,
   Clk => systemCLK,
   PCResult => pc_pcaddim
);

RegF: RegisterFile port map(
   ReadRegister1 => im_conreg(25 downto 21),
   ReadRegister2 => im_conreg(20 downto 16),
   WriteRegister => m1_reg,
```

```
    WriteData => m4_wd,
    RegWrite => con_rw,
    clk => systemCLK,
    ReadData1 => rd1_aluA,
    ReadData2 => rd2_wdm2
);

SigE: SignExtension port map(
    A => im_conreg(15 downto 0),
    SignImm => se_m4sl2
);

PCA: PCAdder port map(
    PCResult => pc_pcaddim,
    PCAddResult => pcadd_adderm3
);

Mux5bits: Mux5bit port map(
    SS => con_rd,
    AA => im_conreg(15 downto 11),
    BB => im_conreg(20 downto 16),
    YY => m1_reg
);

Mux2: Mux port map(
    S => con_alusrc,
    A => se_m4sl2,
    B => rd2_wdm2,
    Y => m2_aluB
);
```

```
Mux3: Mux port map(
   S => con_pcsrc,
   A => adder_m3,
   B => pcadd_adderm3,
   Y => m3_pc
);

Mux4: Mux port map(
   S => con_mtr,
   A => rd_m4,
   B => alu_addm4,
   Y => m4_wd
);

Mux5: seven_MUX port map(
   S => con_alu,
   A => rd1_aluA,
   B => rd2_wdm2,
   Y => alugiris


);

 AdALU: Adder port map(
       X => pcadd_adderm3,
       M => sl2_adder,
       SUM => adder_m3
   );

SL2: shiftleft2 port map(
      entry => se_m4sl2,
```

```
                    exitt => sl2_adder
);

clkdiv1: ClkDiv port map(
      clk => clkIn,
      clkParameter => 50000000,
      clock_out => systemCLK
);
FourDigDisp: FourDigit port map(
BCDin => m4_wd(13 downto 0),
clk => clkIn,
seven_segment => sevenOut,
anode => anodeOut
);

SegOut <= sevenOut;
Anodes <= anodeOut;



end Behavioral;
```

Top Design Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity MIPS_tb is
end;

architecture bench of MIPS_tb is

  component MIPS
```

```
    port(
      clk : in std_logic;
      Reset: in std_logic;
      Mips_out: out std_logic_vector(31 downto 0);
      SegOut: out std_logic_vector(6 downto 0);
      Anodes: out std_logic_vector(3 downto 0)
  );
 end component;

 signal clk_tb: std_logic;
 signal Reset_tb: std_logic ;
 signal Mips_out_tb: std_logic_vector(31 downto 0);
 signal SegOut_tb: std_logic_vector(6 downto 0);
 signal Anodes_tb: std_logic_vector(3 downto 0);

begin

 uut: MIPS port map ( clk   => clk_tb,
                      Reset => Reset_tb,
                      Mips_out => Mips_out_tb,
                      SegOut => SegOut_tb,
                      Anodes => Anodes_tb
                     );
clocking: process
 begin
  clk_tb <= '0';
  wait for 10 ns;
  clk_tb <= '1';
  wait for 10 ns ;
 end process;
```

```
process
begin
Reset_tb <= '1';
wait for 21 ns;
Reset_tb <= '0';
wait for 70 ns;
Reset_tb <= '1';
wait for 21 ns;
Reset_tb <= '0';
wait;
end process;
end;
```

ALU32Bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.NUMERIC_STD.all;

entity ALU32Bit is
  Port (

        A: in std_logic_vector (31 downto 0); --32-Bit input port A.
        B: in std_logic_vector (31 downto 0); --32-Bit input port B.
        ALUControl: in std_logic_vector (3 downto 0); --4-Bit input control bits to select an ALU operation.
        ALUResult: out std_logic_vector (31 downto 0);  --32-Bit ALU result output.
        ZERO: out std_logic --1-Bit output flag.
);
end ALU32Bit;

architecture Behavioral of ALU32Bit is
```

```
signal operationResult: STD_LOGIC_VECTOR(31 DOWNTO 0);
signal shiftleft: STD_LOGIC_VECTOR(31 DOWNTO 0);
signal shiftright: STD_LOGIC_VECTOR(31 DOWNTO 0);
signal shiftAmount: STD_LOGIC_VECTOR(4 DOWNTO 0);


begin

shiftAmount <= B(10 downto 6);
process(shiftAmount)
begin
case shiftAmount is
	when "00000" => shiftleft <= A;
	when "00001" => shiftleft <= A(30 DOWNTO 0) & '0';
	when "00010" => shiftleft <= A(29 DOWNTO 0) & "00";
	when "00011" => shiftleft <= A(28 DOWNTO 0) & "000";
	when "00100" => shiftleft <= A(27 DOWNTO 0) & "0000";
	when "00101" => shiftleft <= A(26 DOWNTO 0) & "00000";
	when "00110" => shiftleft <= A(25 DOWNTO 0) & "000000";
	when "00111" => shiftleft <= A(24 DOWNTO 0) & "0000000";
	when "01000" => shiftleft <= A(23 DOWNTO 0) & "00000000";
	when "01001" => shiftleft <= A(22 DOWNTO 0) & "000000000";
	when "01010" => shiftleft <= A(21 DOWNTO 0) & "0000000000";
	when "01011" => shiftleft <= A(20 DOWNTO 0) & "00000000000";
	when "01100" => shiftleft <= A(19 DOWNTO 0) & "000000000000";
	when "01101" => shiftleft <= A(18 DOWNTO 0) & "0000000000000";
	when "01110" => shiftleft <= A(17 DOWNTO 0) & "00000000000000";
	when "01111" => shiftleft <= A(16 DOWNTO 0) & "000000000000000";
	when "10000" => shiftleft <= A(15 DOWNTO 0) & "0000000000000000";
	when "10001" => shiftleft <= A(14 DOWNTO 0) & "00000000000000000";
	when "10010" => shiftleft <= A(13 DOWNTO 0) & "000000000000000000";
```

```
    when "10011" => shiftleft <= A(12 DOWNTO 0) & "0000000000000000000";
    when "10100" => shiftleft <= A(11 DOWNTO 0) & "00000000000000000000";
    when "10101" => shiftleft <= A(10 DOWNTO 0) & "000000000000000000000";
    when "10110" => shiftleft <= A(9 DOWNTO 0) & "0000000000000000000000";
    when "10111" => shiftleft <= A(8 DOWNTO 0) & "00000000000000000000000";
    when "11000" => shiftleft <= A(7 DOWNTO 0) & "000000000000000000000000";
    when "11001" => shiftleft <= A(6 DOWNTO 0) & "0000000000000000000000000";
    when "11010" => shiftleft <= A(5 DOWNTO 0) & "00000000000000000000000000";
    when "11011" => shiftleft <= A(4 DOWNTO 0) & "000000000000000000000000000";
    when "11100" => shiftleft <= A(3 DOWNTO 0) & "0000000000000000000000000000";
    when "11101" => shiftleft <= A(2 DOWNTO 0) & "00000000000000000000000000000";
    when "11110" => shiftleft <= A(1 DOWNTO 0) & "000000000000000000000000000000";
    when "11111" => shiftleft <= A(0) & "0000000000000000000000000000000";
    when others => shiftleft <= A(0) & "0000000000000000000000000000000";
end case;
case shiftAmount is
    when "00000" => shiftright <= A;
    when "00001" => shiftright <= '0' & A(31 DOWNTO 1);
    when "00010" => shiftright <= "00" & A(31 DOWNTO 2);
    when "00011" => shiftright <= "000" & A(31 DOWNTO 3);
    when "00100" => shiftright <= "0000" & A(31 DOWNTO 4);
    when "00101" => shiftright <= "00000" & A(31 DOWNTO 5);
    when "00110" => shiftright <= "000000" & A(31 DOWNTO 6);
    when "00111" => shiftright <= "0000000" & A(31 DOWNTO 7);
    when "01000" => shiftright <= "00000000" & A(31 DOWNTO 8);
    when "01001" => shiftright <= "000000000" & A(31 DOWNTO 9);
    when "01010" => shiftright <= "0000000000" & A(31 DOWNTO 10);
    when "01011" => shiftright <= "00000000000" & A(31 DOWNTO 11);
    when "01100" => shiftright <= "000000000000" & A(31 DOWNTO 12);
    when "01101" => shiftright <= "0000000000000" & A(31 DOWNTO 13);
    when "01110" => shiftright <= "00000000000000" & A(31 DOWNTO 14);
```

```
        when "01111" => shiftright <= "000000000000000" & A(31 DOWNTO 15);
        when "10000" => shiftright <= "0000000000000000" & A(31 DOWNTO 16);
        when "10001" => shiftright <= "00000000000000000" & A(31 DOWNTO 17);
        when "10010" => shiftright <= "000000000000000000" & A(31 DOWNTO 18);
        when "10011" => shiftright <= "0000000000000000000" & A(31 DOWNTO 19);
        when "10100" => shiftright <= "00000000000000000000" & A(31 DOWNTO 20);
        when "10101" => shiftright <= "000000000000000000000" & A(31 DOWNTO 21);
        when "10110" => shiftright <= "0000000000000000000000" & A(31 DOWNTO 22);
        when "10111" => shiftright <= "00000000000000000000000" & A(31 DOWNTO 23);
        when "11000" => shiftright <= "000000000000000000000000" & A(31 DOWNTO 24);
        when "11001" => shiftright <= "0000000000000000000000000" & A(31 DOWNTO 25);
        when "11010" => shiftright <= "00000000000000000000000000" & A(31 DOWNTO 26);
        when "11011" => shiftright <= "000000000000000000000000000" & A(31 DOWNTO 27);
        when "11100" => shiftright <= "0000000000000000000000000000" & A(31 DOWNTO 28);
        when "11101" => shiftright <= "00000000000000000000000000000" & A(31 DOWNTO 29);
        when "11110" => shiftright <= "000000000000000000000000000000" & A(31 DOWNTO 30);
        when "11111" => shiftright <= "0000000000000000000000000000000" & A(31);
        when others => shiftright <=  "00000000000000000000000000000000";
    end case;
end process;


   process(A,B, ALUControl)
         variable count : unsigned(31 downto 0) := x"00000000";
         variable z: STD_LOGIC_VECTOR(63 DOWNTO 0):= x"0000000000000000";
         begin
                case ALUControl is
                        when "0000" => operationResult <= A + B; --addition
                        when "0001" => operationResult <= A - B; --subtraction
                        when "0010" => z := (A * B); --multiplication
          operationResult <= z(31 downto 0);
```

```
                        when "0011" => operationResult <= A and B; --and
                        when "0100" => operationResult <= A or B; --or
                        when "0101" => -- Less than comparison
            if(A<B) then
             operationResult <= x"00000001" ;
            else
             operationResult <= x"00000000" ;
            end if;
                        when "0110" => -- Equal comparison
            if(A=B) then
             operationResult <= x"00000001" ;
            else
             operationResult <= x"00000000" ;
            end if;
                        when "0111" => -- Not equal comparison
            if(A /= B) then
             operationResult <= x"00000001" ;
            else
             operationResult <= x"00000000" ;
            end if;
                        when "1000" => operationResult <= shiftleft;
                        when "1001" => operationResult <= shiftright;
                        when "1010" => operationResult <= std_logic_vector(unsigned(A) ror
to_integer(unsigned(B)));
          when "1011" =>
            count := x"00000000";
            for i in 0 to 31 loop   --check for all the bits.
               if(A(i) = '1') then --check if the bit is '1'
                  count := count + 1; --if its one, increment the count.
               end if;
            end loop;
```

```
            operationResult <= std_logic_vector(count);
          when "1100" =>
            count := x"00000000";
            for i in 0 to 31 loop   --check for all the bits.
              if(A(i) = '0') then --check if the bit is '0'
                 count := count + 1; --if its zero, increment the count.
              end if;
            end loop;
            operationResult <= std_logic_vector(count);
                              when others => NULL;
                    end case;

           end process;

process(operationResult)
begin
                    if operationResult = 0 then
                      ZERO <= '1';
                    else
                      ZERO <= '0';
                    end if;

end process;

ALUResult <= operationResult;
end Behavioral;
```

ALU32Bit_tb:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_unsigned.all;
```

```
ENTITY ALU32Bit_tb IS

END ALU32Bit_tb;


ARCHITECTURE Behavioral OF ALU32Bit_tb IS

   COMPONENT ALU32Bit

   PORT(

         A: in std_logic_vector (31 downto 0); --32-Bit input port A.

         B: in std_logic_vector (31 downto 0); --32-Bit input port B.

         ALUControl: in std_logic_vector (3 downto 0); --4-Bit input control bits to select an ALU
operation.

-- OUTPUTS:

         ALUResult: out std_logic_vector (31 downto 0);  --32-Bit ALU result output.

         ZERO: out std_logic --1-Bit output flag.

       );

   END COMPONENT;

      --Inputs

  signal A : std_logic_vector(31 downto 0) := (others => '0');

  signal B : std_logic_vector(31 downto 0) := (others => '0');

  signal ALUControl : std_logic_vector(3 downto 0) := (others => '0');

 --Outputs

  signal ALUResult : std_logic_vector(31 downto 0);

  signal ZERO : std_logic;

BEGIN

  uut: ALU32Bit PORT MAP (

        A => A,

        B => B,

        ALUControl => ALUControl,

        ALUResult => ALUResult,

        ZERO => ZERO

      );

  stim_proc: process
```

```
  begin
   A <="00010101010011110000001111101001";--00010101010011110000001111101001
   B <="00000000000000000000000000000101";
   ALUControl <= "0000";
   wait for 10 ns;
   ALUControl <= "0001";
   wait for 10 ns;
   ALUControl <= "0010";
   wait for 10 ns;
   ALUControl <= "0011";
   wait for 10 ns;
   ALUControl <= "0100";
   wait for 10 ns;
   ALUControl <= "0101";
   wait for 10 ns;
   ALUControl <= "0110";
   wait for 10 ns;
   ALUControl <= "0111";
   wait for 10 ns;
   ALUControl <= "1000";
   wait for 10 ns;
   ALUControl <= "1001";
   wait for 10 ns;
   ALUControl <= "1010";
   wait for 10 ns;
   ALUControl <= "1011";
   wait for 10 ns;
   ALUControl <= "1100";
   wait;
  end process;
END;
```

Instruction Memory:

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

entity instructionmemory is
---------//ports//-------------------------------------
PORT (address: in STD_LOGIC_VECTOR(31 DOWNTO 0);
            instruction: out STD_LOGIC_VECTOR(31 DOWNTO 0));
-----------------------------------------------
end instructionmemory;

architecture Behavioralx of instructionmemory is
--------//signals//------------------------
  type vector_of_mem is array(0 to 127) of std_logic_vector (31 downto 0);
  signal memory : vector_of_mem := (
"00100000000100000000000000001110", --x"2010000e",
"00100000000100010000000000001111", --x"2011000f",
"00100000000100100000000000011101",        --x"2012001d",
"00100000000100111111111111110001", --x"2013fff1",
"00000010001100100100000000100000", --x"02324020", --00000010001100100100000000100000

"00000010010100000100000000100100",--memory(5) = x"02504024", --
"01110010000100010100000000000010",--memory(6) = x"72114002",
"00000010010100000100000000100101",--memory(7) = x"02504025",
"00110110000010000000000000010000",--memory(8) = x"36080010",
"00000010000100100100000000100010",--memory(9) = x"02124022",
"01110010011000000100000000100001",--memory(10) = x"72604021",
"01110010010000000100000000100000",--memory(11) = x"72404020",
"00000010000100010100000000101010",--memory(12) = x"0211402a",
"00000010001100000100000000101010",--memory(13) = x"0230402a",
```

```
"00000000000100010100000010000000",--memory(14) = x"00114080",
"00000000000100100100000011000010",--memory(15) = x"001240c2"
others => "00000000000000000000000000000000"
 );

begin
   instruction <= memory(CONV_INTEGER(address(8 downto 2)));
--------------------------------------------------------------------------------
end Behavioralx;
```

Program Counter:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ProgramCounter IS
PORT(
Address: IN STD_LOGIC_VECTOR(31 downto 0); -- input
Reset : IN STD_LOGIC; -- async. clear.
Clk : IN STD_LOGIC; -- clock.
PCResult: OUT STD_LOGIC_VECTOR(31 downto 0) -- output.
);
END ProgramCounter;

ARCHITECTURE Behavioral of ProgramCounter is
BEGIN
process(Clk, Reset)
begin
if rising_edge(Clk) then
   if Reset = '1' then
   PCResult <= x"00000000";
   else -- rising_edge(Clk) then
   PCResult <= Address;
   end if;
```

```
end if;
end process;
end Behavioral;
```

Program Counter Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity ProgramCounter_tb is
end;
architecture Behavioral of ProgramCounter_tb is
 component ProgramCounter is
 PORT(
 Address: IN STD_LOGIC_VECTOR(31 downto 0):=x"00000000";
 Reset : IN STD_LOGIC;
 Clk : IN STD_LOGIC;
 PCResult: OUT STD_LOGIC_VECTOR(31 downto 0)
 );
 end component;

 signal Address: STD_LOGIC_VECTOR(31 downto 0);
 signal Reset: STD_LOGIC;
 signal Clk: STD_LOGIC;
 signal PCResult: STD_LOGIC_VECTOR(31 downto 0) ;

 constant clock_period: time := 5 ns;
begin
 uut: ProgramCounter port map ( Address  => Address,
                Reset    => Reset,
                Clk      => Clk,
                PCResult => PCResult );
```

```
 stimulus: process
 begin
 Clk <= '0';
 wait for clock_period*2;
 Clk <= '1';
 wait for clock_period*2;
 end process;

process
begin
   Address <= "11111000001111100000111110000011";
   Reset <= '0';
   wait for clock_period*5;
   Reset <= '1';
   wait for clock_period*4;
   Reset <= '0';
   wait;
 end process;
end Behavioral;
```

Register File:

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

entity RegisterFile is
port(
   ReadRegister1: in std_logic_vector(4 downto 0);
   ReadRegister2: in std_logic_vector(4 downto 0);
   WriteRegister: in std_logic_vector(4 downto 0);
   WriteData: in std_logic_vector(31 downto 0);
   RegWrite: in std_logic;
   clk: in std_logic;
```

```
    ReadData1: out std_logic_vector(31 downto 0):=x"00000000";
    ReadData2: out std_logic_vector(31 downto 0):=x"00000000"
);
end RegisterFile;

architecture Behavioral of RegisterFile is

signal register0: std_logic_vector(31 downto 0):= x"00000000";
signal register1: std_logic_vector(31 downto 0):= x"00000000";
signal register2: std_logic_vector(31 downto 0):= x"00000000";
signal register3: std_logic_vector(31 downto 0):= x"00000000";
signal register4: std_logic_vector(31 downto 0):= x"00000000";
signal register5: std_logic_vector(31 downto 0):= x"00000000";
signal register6: std_logic_vector(31 downto 0):= x"00000000";
signal register7: std_logic_vector(31 downto 0):= x"00000000";
signal register8: std_logic_vector(31 downto 0):= x"00000000";
signal register9: std_logic_vector(31 downto 0):= x"00000000";
signal register10: std_logic_vector(31 downto 0):= x"00000000";
signal register11: std_logic_vector(31 downto 0):= x"00000000";
signal register12: std_logic_vector(31 downto 0):= x"00000000";
signal register13: std_logic_vector(31 downto 0):= x"00000000";
signal register14: std_logic_vector(31 downto 0):= x"00000000";
signal register15: std_logic_vector(31 downto 0):= x"00000000";
signal register16: std_logic_vector(31 downto 0):= x"00000000";
signal register17: std_logic_vector(31 downto 0):= x"00000000";
signal register18: std_logic_vector(31 downto 0):= x"00000000";
signal register19: std_logic_vector(31 downto 0):= x"00000000";
signal register20: std_logic_vector(31 downto 0):= x"00000000";
signal register21: std_logic_vector(31 downto 0):= x"00000000";
signal register22: std_logic_vector(31 downto 0):= x"00000000";
```

```
signal register23: std_logic_vector(31 downto 0):= x"00000000";
signal register24: std_logic_vector(31 downto 0):= x"00000000";
signal register25: std_logic_vector(31 downto 0):= x"00000000";
signal register26: std_logic_vector(31 downto 0):= x"00000000";
signal register27: std_logic_vector(31 downto 0):= x"00000000";
signal register28: std_logic_vector(31 downto 0):= x"00000000";
signal register29: std_logic_vector(31 downto 0):= x"00000000";
signal register30: std_logic_vector(31 downto 0):= x"00000000";
signal register31: std_logic_vector(31 downto 0):= x"00000000";

begin

   process(clk, WriteRegister)
   begin
      if RegWrite = '1' then
         if rising_edge(clk) then
            case WriteRegister is
               when "00000" => register0 <= WriteData;
               when "00001" => register1 <= WriteData;
               when "00010" => register2 <= WriteData;
               when "00011" => register3 <= WriteData;
               when "00100" => register4 <= WriteData;
               when "00101" => register5 <= WriteData;
               when "00110" => register6 <= WriteData;
               when "00111" => register7 <= WriteData;
               when "01000" => register8 <= WriteData;
               when "01001" => register9 <= WriteData;
               when "01010" => register10 <= WriteData;
               when "01011" => register11 <= WriteData;
               when "01100" => register12 <= WriteData;
               when "01101" => register13 <= WriteData;
```

```
            when "01110" => register14 <= WriteData;
            when "01111" => register15 <= WriteData;
            when "10000" => register16 <= WriteData;
            when "10001" => register17 <= WriteData;
            when "10010" => register18 <= WriteData;
            when "10011" => register19 <= WriteData;
            when "10100" => register20 <= WriteData;
            when "10101" => register21 <= WriteData;
            when "10110" => register22 <= WriteData;
            when "10111" => register23 <= WriteData;
            when "11000" => register24 <= WriteData;
            when "11001" => register25 <= WriteData;
            when "11010" => register26 <= WriteData;
            when "11011" => register27 <= WriteData;
            when "11100" => register28 <= WriteData;
            when "11101" => register29 <= WriteData;
            when "11110" => register30 <= WriteData;
            when "11111" => register31 <= WriteData;
            when others => register31 <= WriteData;
        end case;
   end if;
  end if;
end process;

process(clk,ReadRegister1, ReadRegister2)
begin
   if falling_edge(clk) then
      case ReadRegister1 is
         when "00000" => ReadData1 <= register0 ;
         when "00001" => ReadData1 <= register1 ;
         when "00010" => ReadData1 <= register2 ;
```

```
    when "00011" => ReadData1 <= register3 ;
    when "00100" => ReadData1 <= register4 ;
    when "00101" => ReadData1 <= register5 ;
    when "00110" => ReadData1 <= register6 ;
    when "00111" => ReadData1 <= register7 ;
    when "01000" => ReadData1 <= register8 ;
    when "01001" => ReadData1 <= register9 ;
    when "01010" => ReadData1 <= register10 ;
    when "01011" => ReadData1 <= register11 ;
    when "01100" => ReadData1 <= register12 ;
    when "01101" => ReadData1 <= register13 ;
    when "01110" => ReadData1 <= register14 ;
    when "01111" => ReadData1 <= register15 ;
    when "10000" => ReadData1 <= register16 ;
    when "10001" => ReadData1 <= register17 ;
    when "10010" => ReadData1 <= register18 ;
    when "10011" => ReadData1 <= register19 ;
    when "10100" => ReadData1 <= register20 ;
    when "10101" => ReadData1 <= register21 ;
    when "10110" => ReadData1 <= register22 ;
    when "10111" => ReadData1 <= register23 ;
    when "11000" => ReadData1 <= register24 ;
    when "11001" => ReadData1 <= register25 ;
    when "11010" => ReadData1 <= register26 ;
    when "11011" => ReadData1 <= register27 ;
    when "11100" => ReadData1 <= register28 ;
    when "11101" => ReadData1 <= register29 ;
    when "11110" => ReadData1 <= register30 ;
    when "11111" => ReadData1 <= register31 ;
    when others => ReadData1 <= register31 ;
end case;
```

```
case ReadRegister2 is
    when "00000" => ReadData2 <= register0 ;
    when "00001" => ReadData2 <= register1 ;
    when "00010" => ReadData2 <= register2 ;
    when "00011" => ReadData2 <= register3 ;
    when "00100" => ReadData2 <= register4 ;
    when "00101" => ReadData2 <= register5 ;
    when "00110" => ReadData2 <= register6 ;
    when "00111" => ReadData2 <= register7 ;
    when "01000" => ReadData2 <= register8 ;
    when "01001" => ReadData2 <= register9 ;
    when "01010" => ReadData2 <= register10 ;
    when "01011" => ReadData2 <= register11 ;
    when "01100" => ReadData2 <= register12 ;
    when "01101" => ReadData2 <= register13 ;
    when "01110" => ReadData2 <= register14 ;
    when "01111" => ReadData2 <= register15 ;
    when "10000" => ReadData2 <= register16 ;
    when "10001" => ReadData2 <= register17 ;
    when "10010" => ReadData2 <= register18 ;
    when "10011" => ReadData2 <= register19 ;
    when "10100" => ReadData2 <= register20 ;
    when "10101" => ReadData2 <= register21 ;
    when "10110" => ReadData2 <= register22 ;
    when "10111" => ReadData2 <= register23 ;
    when "11000" => ReadData2 <= register24 ;
    when "11001" => ReadData2 <= register25 ;
    when "11010" => ReadData2 <= register26 ;
    when "11011" => ReadData2 <= register27 ;
    when "11100" => ReadData2 <= register28 ;
    when "11101" => ReadData2 <= register29 ;
```

```
            when "11110" => ReadData2 <= register30 ;
            when "11111" => ReadData2 <= register31 ;
            when others => ReadData2 <= register31 ;
        end case;
    end if;
  end process;
end Behavioral;
```

Register File Testbench:

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

entity RegisterFile_tb is
end RegisterFile_tb;

architecture Behavioral of RegisterFile_tb is

signal ReadRegister1_tb: std_logic_vector(4 downto 0);
signal ReadRegister2_tb: std_logic_vector(4 downto 0);
signal WriteRegister_tb: std_logic_vector(4 downto 0);
signal WriteData_tb: std_logic_vector(31 downto 0);
signal RegWrite_tb: std_logic;
signal clk_tb: std_logic;
signal ReadData1_tb: std_logic_vector(31 downto 0);
signal ReadData2_tb: std_logic_vector(31 downto 0);

component RegisterFile is
port(
  ReadRegister1: in std_logic_vector(4 downto 0);
  ReadRegister2: in std_logic_vector(4 downto 0);
  WriteRegister: in std_logic_vector(4 downto 0);
  WriteData: in std_logic_vector(31 downto 0);
```

```
    RegWrite: in std_logic;
    clk: in std_logic;
    ReadData1: out std_logic_vector(31 downto 0);
    ReadData2: out std_logic_vector(31 downto 0)
);
end component;
begin
UUT: RegisterFile port map(
    ReadRegister1 => ReadRegister1_tb,
    ReadRegister2 => ReadRegister2_tb,
    WriteRegister => WriteRegister_tb,
    WriteData => WriteData_tb,
    RegWrite => RegWrite_tb,
    clk => clk_tb,
    ReadData1 => ReadData1_tb,
    ReadData2 => ReadData2_tb
);
----------------
  process
  begin
    clk_tb <= '0';
    wait for 10 ns;
    clk_tb <= '1';
    wait for 10 ns ;
  end process;
 ---------------
 process
 begin
    ReadRegister1_tb <= "00011";
    ReadRegister2_tb <= "00100";
    WriteData_tb <= "00000000001100000000000000001101";
```

```
   WriteRegister_tb <= "00000";
   RegWrite_tb <= '0';
   wait for 25 ns;
   RegWrite_tb <= '1';
   wait for 20 ns;
   WriteRegister_tb <= "00011";
   wait for 20 ns;
   WriteRegister_tb <= "00100";
   wait;
 end process;
end Behavioral;
```

Sign Extension:

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

entity SignExtension is

PORT (
A: in STD_LOGIC_VECTOR(15 DOWNTO 0);
SignImm: out STD_LOGIC_VECTOR(31 DOWNTO 0)
);
end SignExtension;

architecture Behavioral of SignExtension is
begin
process(A)
begin
   if (A(15) = '0') then
      SignImm(15 downto 0)<=A(15 downto 0);
      SignImm(31 downto 16)<="0000000000000000";
   else
```

```
      SignImm(15 downto 0)<=A(15 downto 0);
      SignImm(31 downto 16)<="1111111111111111";
   end if;
end process;
end Behavioral;
```

Sign Extension Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity SignExtension_tb is
end;

architecture behavioral of SignExtension_tb is

 component SignExtension
 PORT (
 A: in STD_LOGIC_VECTOR(15 DOWNTO 0);
 SignImm: out STD_LOGIC_VECTOR(31 DOWNTO 0));
 end component;

 signal A: STD_LOGIC_VECTOR(15 DOWNTO 0);
 signal SignImm: STD_LOGIC_VECTOR(31 DOWNTO 0);

begin
 uut: SignExtension port map (
   A      => A,
   SignImm => SignImm
   );
process
 begin
```

```
   A <= "0010101010101010";
   wait for 10 ns;
   A <= "1010101010101010";
   wait;
end process;
end;
```

PC Adder:

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

entity PCAdder is
port(
   PCResult: in std_logic_vector (31 downto 0);
   PCAddResult: out std_logic_vector(31 downto 0)
);
end PCAdder;

architecture Behavioral of PCAdder is
begin
   PCAddResult <= PCResult + "100";
end Behavioral;
```

PC Adder Testbench:

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

entity PCAdder_tb is
end PCAdder_tb;

architecture Behavioral of PCAdder_tb is
signal PCResult_tb: std_logic_vector (31 downto 0);
```

```
signal PCAddResult_tb: std_logic_vector (31 downto 0);


component PCAdder is
   port(
      PCResult: in std_logic_vector (31 downto 0);
      PCAddResult: out std_logic_vector(31 downto 0)
   );
end component;
begin
UUT: PCAdder port map(
   PCResult => PCResult_tb,
   PCAddResult => PCAddResult_tb
);
process begin
   PCResult_tb <= "11111111111111111111111111111100";
   wait for 10 ns;
   PCResult_tb <= "00000000000000000000000000100000";
   wait for 10 ns;
   PCResult_tb <= "00001010001111000110000000100000";
   wait for 10 ns;
   PCResult_tb <= "00100110110000000000111111010100";
   wait;
end process;
end Behavioral;
```

MUX:

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


entity Mux is
   Port ( S : in STD_LOGIC;
          A : in STD_LOGIC_VECTOR (31 downto 0);
```

```
        B : in STD_LOGIC_VECTOR (31 downto 0);
        Y : out STD_LOGIC_VECTOR (31 downto 0));
end Mux;

architecture Behavioral of Mux is
begin
   process(A, B, S)
   begin
   if(S = '1') then
      Y <= A;
   else
      Y <= B;
   end if;
   end process;
end Behavioral;
```

MUX Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity Mux_tb is
end;

architecture bench of Mux_tb is

  component Mux
     Port ( S : in STD_LOGIC;
          A : in STD_LOGIC_VECTOR (31 downto 0);
          B : in STD_LOGIC_VECTOR (31 downto 0);
          Y : out STD_LOGIC_VECTOR (31 downto 0));
  end component;
```

```
 signal S: STD_LOGIC;
 signal A: STD_LOGIC_VECTOR (31 downto 0);
 signal B: STD_LOGIC_VECTOR (31 downto 0);
 signal Y: STD_LOGIC_VECTOR (31 downto 0);

begin

uut: Mux port map ( S => S,
                    A => A,
                    B => B,
                    Y => Y );
stimulus_d: process
 begin
 A<="00000000011100011100011100011101";
 B<="00000000000100001111000011110000";
 wait;
 end process;
stimulus_s: process
 begin
S<='0';
 wait for 50 ns;
S<='1';
 wait for 50 ns;
 end process;
end;
```

Data Memory:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```
entity dataMemory is
  port(
    clk  : in std_logic;
    memWrite  : in std_logic;
    memRead   : in std_logic;
    address      : in std_logic_vector(31 downto 0);
    writeData     : in std_logic_vector(31 downto 0);
    readData      : out std_logic_vector(31 downto 0)
  );
end dataMemory;

architecture behavioral of dataMemory is
  type vector_of_mem is array(0 to 63) of std_logic_vector (31 downto 0);
  signal memory : vector_of_mem := (
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
       "00000000000000000000000000000000",
```

"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",
"00000000000000000000000000000000",

```
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000",
      "00000000000000000000000000000000"
   );
 begin


  readData <= memory(to_integer(unsigned(address(7 downto 2)))) when memRead = '1' else
x"00000000"; -- we took address as 6 bit because of 64 bit


  operation : process(memWrite, address, writeData)
   begin
   if (clk'EVENT AND clk='0') then
     if memWrite = '1' then
       memory(to_integer(unsigned(address(7 downto 2)))) <= writeData;
     end if;
   end if;
 end process operation;
end behavioral;
```

Data Memory Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity dataMemory_tb is
end;

architecture bench of dataMemory_tb is
 component dataMemory
  port(
   clk  : in std_logic;
   memWrite  : in std_logic;
   memRead   : in std_logic;
   address      : in std_logic_vector(31 downto 0);
   writeData    : in std_logic_vector(31 downto 0);
   readData     : out std_logic_vector(31 downto 0)
  );
 end component;

 signal clk_tb: std_logic;
 signal memWrite_tb: std_logic;
 signal memRead_tb: std_logic;
 signal address_tb: std_logic_vector(31 downto 0);
 signal writeData_tb: std_logic_vector(31 downto 0);
 signal readData_tb: std_logic_vector(31 downto 0) ;

begin
 uut: dataMemory port map ( clk      => clk_tb,
                    memWrite  => memWrite_tb,
                    memRead   => memRead_tb,
```

```
                address   => address_tb,
                writeData => writeData_tb,
                readData  => readData_tb );

 clocking: process
 begin
   clk_tb <= '0';
   wait for 10 ns;
   clk_tb <= '1';
   wait for 10 ns ;
 end process;

process
 begin
   memRead_tb <= '0';
   address_tb   <= "00000000000000000000000000000101";
          writeData_tb <= x"11112222";
          memWrite_tb  <= '0';
          WAIT FOR 5 ns;
          memWrite_tb <= '1';
          WAIT FOR 10 ns;
          memWrite_tb <= '0';
          WAIT FOR 10 ns;
          address_tb   <= "00000000000000000000000000001101";
          writeData_tb <= x"33334444";
          memWrite_tb  <= '0';
          WAIT FOR 10 ns;
          memWrite_tb <= '1';
          WAIT FOR 10 ns;
          memWrite_tb <= '0';
          WAIT FOR 10 ns;
```

```
        -- read two memory locations
        address_tb <= "00000000000000000000000000000101";
        memRead_tb <= '0';
        WAIT FOR 10 ns;
        memRead_tb <= '1';
        WAIT FOR 10 ns;
        memRead_tb <= '0';
        WAIT FOR 10 ns;
        address_tb <= "00000000000000000000000000001101";
        memRead_tb <= '0';
        WAIT FOR 10 ns;
        memRead_tb <= '1';
        WAIT FOR 10 ns;
        memRead_tb <= '0';
        wait;
 end process;
end;
```

Controller:

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

entity Controller is
---------//ports//---------
PORT (
Op: in STD_LOGIC_VECTOR(5 DOWNTO 0);
Funct: in STD_LOGIC_VECTOR(5 DOWNTO 0);
MemRead: out std_logic;
MemtoReg: out STD_LOGIC;
MemWrite: out STD_LOGIC;
ALUControl: out STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
ALUSrc: out STD_LOGIC;
RegDst: out STD_LOGIC;
RegWrite: out STD_LOGIC;
PCSrc: out std_logic
);
--------------------------
end Controller;

architecture Behavioral of Controller is
signal outputVector: std_logic_vector(6 downto 0);

begin
process(Op, Funct)
begin
case Op is
   when "000000" => case Funct is
                when "100000" => ALUControl <= "0000"; -- add
                        outputVector <= "0110000";
                when "100010" => ALUControl <= "0001"; -- sub
                        outputVector <= "0110000";
                when "100100" => ALUControl <= "0011";-- and
                        outputVector <= "0110000";
                when "100101" => ALUControl <= "0100"; -- or
                        outputVector <= "0110000";
                when "101010" => ALUControl <= "0101"; -- set less than
                        outputVector <= "0110000";
                when "111001" => ALUControl <= "0110"; -- set equal
                        outputVector <= "0110000";
                when "111010" => ALUControl <= "0111"; -- set not equal than
                        outputVector <= "0110000";
                when "000000" => ALUControl <= "1000"; -- shift left
```

```
                    outputVector <= "1110000";
                when "000010" => ALUControl <= "1001"; -- shift right
                    outputVector <= "1110000";
                when "000110" => ALUControl <= "1010"; -- rotate right
                    outputVector <= "0110000"; --ilk bite bir daha bak
                when others => ALUControl <= "0000";
                    outputVector <= "0110000";
            end case;
    when "011100" => case Funct is
                when "100001" => ALUControl <= "1011"; -- clo
                    outputVector <= "0110000";
                when "100000" => ALUControl <= "1100"; -- clz
                    outputVector <= "0110000";
                when "000010" => ALUControl <= "0010"; -- mul
                    outputVector <= "0110000";
                when others => ALUControl <= "0000";
                    outputVector <= "0110000";
            end case;
    when "001000" => ALUControl <= "0000"; -- add immediate
            outputVector <= "1010000";
    when "001101" => ALUControl <= "0100"; -- or immediate
            outputVector <= "1010000";
    when others => ALUControl <="0000";
            outputVector <= "0110000";
end case;
end process;
ALUSrc <= outputVector(6);
RegDst <= outputVector(5);
RegWrite <= outputVector(4);
MemRead <= outputVector(3);
MemWrite <= outputVector(2);
```

```
MemtoReg <= outputVector(1);
PCSrc <= outputVector(0);
end Behavioral;
```

Controller Testbench:

```
library IEEE;
use IEEE.Std_logic_1164.all;
use IEEE.Numeric_Std.all;

entity Controller_tb is
end;

architecture bench of Controller_tb is
 component Controller
 PORT (
 Op: in STD_LOGIC_VECTOR(5 DOWNTO 0);
 Funct: in STD_LOGIC_VECTOR(5 DOWNTO 0);
 MemRead: out std_logic;
 MemtoReg: out STD_LOGIC;
 MemWrite: out STD_LOGIC;
 ALUControl: out STD_LOGIC_VECTOR(3 DOWNTO 0);
 ALUSrc: out STD_LOGIC;
 RegDst: out STD_LOGIC;
 RegWrite: out STD_LOGIC;
 PCSrc: out std_logic
 );
 end component;

 signal Op: STD_LOGIC_VECTOR(5 DOWNTO 0);
 signal Funct: STD_LOGIC_VECTOR(5 DOWNTO 0);
 signal MemRead: std_logic;
 signal MemtoReg: STD_LOGIC;
```

```
  signal MemWrite: STD_LOGIC;
  signal ALUControl: STD_LOGIC_VECTOR(3 DOWNTO 0);
  signal ALUSrc: STD_LOGIC;
  signal RegDst: STD_LOGIC;
  signal RegWrite: STD_LOGIC;
  signal PCSrc: std_logic ;

begin
uut: Controller port map ( Op        => Op,
                           Funct     => Funct,
                           MemRead   => MemRead,
                           MemtoReg  => MemtoReg,
                           MemWrite  => MemWrite,
                           ALUControl => ALUControl,
                           ALUSrc    => ALUSrc,
                           RegDst    => RegDst,
                           RegWrite  => RegWrite,
                           PCSrc     => PCSrc );
stimulus: process
  begin
  Op <= "000000";
  Funct <= "100000";
  wait for 10 ns;
  Funct <= "100010";
  wait for 10 ns;
  Funct <= "100100";
  wait for 10 ns;
  Funct <= "100101";
  wait for 10 ns;
  Funct <= "101010";
  wait for 10 ns;
```

```
 Funct <= "111001";
 wait for 10 ns;
 Funct <= "111010";
 wait for 10 ns;
 Funct <= "000000";
 wait for 10 ns;
 Funct <= "000010";
 wait for 10 ns;
 Op <= "011100";
 Funct <= "100001";
 wait for 10 ns;
 Funct <= "100000";
 wait for 10 ns;
 Funct <= "000010";
 wait for 10 ns;
 Op <= "001000";
 wait for 10 ns;
 Op <= "001101";
 wait for 10 ns;
 Op <= "000000";
 Funct <= "000110";
 wait;
 end process;
end;
```

Four Digit Display:

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

entity FourDigit is
Port (
BCDin: in std_logic_vector (13 downto 0);
```

```
clk: in std_logic;
seven_segment : out std_logic_vector (6 downto 0);
anode: out std_logic_vector (3 downto 0)
);
end FourDigit;

architecture Behavioral of FourDigit is
signal count: natural range 0 to 100000 :=0;
signal an_number: natural range 0 to 3 :=0;
begin
process(clk)
begin
if(rising_edge(clk)) then
count <=count+1;
if(count >= 100000) then
count <=0;
an_number <= an_number +1;
if (an_number>3) then
an_number <=0;
end if;
end if;
end if;
end process;

process(an_number)
begin
   case an_number is
   when 0 => anode <= "1110";
   when 1 => anode <= "1101";
          when 2 => anode <= "1011";
   when 3 => anode <= "0111";
```

```
   when others => anode <= "1111";
end case;
end process;

process(an_number)
begin
case an_number is
when 0=>
case BCDin is
when "00000000000000" => seven_segment <= "1000000"; --0
when "00000000000001" => seven_segment <= "1111001"; --1
when "00000000000011" => seven_segment <= "0110000"; --3
when "00000000011100" => seven_segment <= "0000000"; --28 sekizi yazdır
when "00000000011101" => seven_segment <= "0010000"; --29 dokuzu yazdır
when "00000000001100" => seven_segment <= "0100100";--12 ikiyi yazdır
when "00000000001101" => seven_segment <= "0110000";--13 üçü yazıdr
when "00000000001110" => seven_segment <= "0011001";--14 dördü yazdır
when "00000000001111" => seven_segment <= "0010010";--15 beşi yazdır
when "00000110010110" => seven_segment <= "0000010"; --406 altıyı yazdır yazdır
when "00000000101100" => seven_segment <= "0011001";--44 dördü yazdır
when "00000011010010" => seven_segment <= "1000000";--210 sıfırı yazdır
when "00000000011111" => seven_segment <= "1111001";--31 biri yazdır
when "00000000011110" => seven_segment <= "1000000";--30 sıfırı yazdır
when "00000000111100" => seven_segment <= "1000000";--60 sıfırı yazdır
when "11111111110001" => seven_segment <= "0010000"; --16369 dokuzu yazdır
when "11111111101111" => seven_segment <= "1111000";--16367 altıyı yazdır
when others => seven_segment <= "1111111";
end case;
when 1=>
case BCDin is
when "00000000000000" => seven_segment <= "1000000"; --0 sıfır yazdır
```

```
when "00000000000001" => seven_segment <= "1000000"; --1 sıfır yazdır
when "00000000000011" => seven_segment <= "1000000"; --3 sıfır yazdır
when "00000000011100" => seven_segment <= "0100100"; --28 ikiyi yazdır
when "00000000011101" => seven_segment <= "0100100"; --29 ikiyi yazdır
when "00000000001100" => seven_segment <= "1111001";--12 biri yazdır
when "00000000001101" => seven_segment <= "1111001";--13 biri yazıdr
when "00000000001110" => seven_segment <= "1111001";--14 biri yazdır
when "00000000001111" => seven_segment <= "1111001";--15 biri yazdır
when "00000110010110" => seven_segment <= "1000000"; --406 sıfırı yazdır yazdır
when "00000000101100" => seven_segment <= "0011001";--44 dördü yazdır
when "00000011010010" => seven_segment <= "1111001";--210 biri yazdır
when "00000000011111" => seven_segment <= "0110000";--31 üçü yazdır
when "00000000011110" => seven_segment <= "0110000";--30 üçü yazdır
when "00000000111100" => seven_segment <= "0000010";--60 altıyı yazdır
when "11111111110001" => seven_segment <= "0000010"; --16369 altı yazdır
when "11111111101111" => seven_segment <= "0000010";--16367 altıyı yazdır
when others => seven_segment <= "1111111";
end case;
when 2=>
case BCDin is
when "00000000000000" => seven_segment <= "1000000"; --0 sıfır yazdır
when "00000000000001" => seven_segment <= "1000000"; --1 sıfır yazdır
when "00000000000011" => seven_segment <= "1000000"; --3 sıfır yazdır
when "00000000011100" => seven_segment <= "1000000"; --28 sıfır yazdır
when "00000000011101" => seven_segment <= "1000000"; --29 sıfır yazdır
when "00000000001100" => seven_segment <= "1000000";--12 sıfır yazdır
when "00000000001101" => seven_segment <= "1000000";--13 sıfır yazıdr
when "00000000001110" => seven_segment <= "1000000";--14 sıfır yazdır
when "00000000001111" => seven_segment <= "1000000";--15 sıfır yazdır
when "00000110010110" => seven_segment <= "0011001"; --406 dördü yazdır
when "00000000101100" => seven_segment <= "1000000";--44 sıfır yazdır
```

```
when "00000011010010" => seven_segment <= "0100100";--210 sıfır yazdır
when "00000000011111" => seven_segment <= "1000000";--31 sıfır yazdır
when "00000000011110" => seven_segment <= "1000000";--30 sıfır yazdır
when "00000000111100" => seven_segment <= "1000000";--60 sıfır yazdır
when "11111111110001" => seven_segment <= "0110000"; --16369 üç yazdır
when "11111111101111" => seven_segment <= "0110000";--16367 altıyı yazdır
when others => seven_segment <= "1111111";
end case;
when 3 =>
case BCDin is
when "00000000000000" => seven_segment <= "1000000"; --0 sıfır yazdır
when "00000000000001" => seven_segment <= "1000000"; --1 sıfır yazdır
when "00000000000011" => seven_segment <= "1000000"; --3 sıfır yazdır
when "00000000011100" => seven_segment <= "1000000"; --28 sıfır yazdır
when "00000000011101" => seven_segment <= "1000000"; --29 sıfır yazdır
when "00000000001100" => seven_segment <= "1000000";--12 sıfır yazdır
when "00000000001101" => seven_segment <= "1000000";--13 sıfır yazıdr
when "00000000001110" => seven_segment <= "1000000";--14 sıfır yazdır
when "00000000001111" => seven_segment <= "1000000";--15 sıfır yazdır
when "00000110010110" => seven_segment <= "1000000"; --406 sıfır yazdır
when "00000000101100" => seven_segment <= "1000000";--44 sıfır yazdır
when "00000011010010" => seven_segment <= "1000000";--210 sıfır yazdır
when "00000000011111" => seven_segment <= "1000000";--31 sıfır yazdır
when "00000000011110" => seven_segment <= "1000000";--30 sıfır yazdır
when "00000000111100" => seven_segment <= "1000000";--60 sıfır yazdır
when "11111111110001" => seven_segment <= "0000010";--16369 altıyı yazdır
when "11111111101111" => seven_segment <= "0000010";--16367 altıyı yazdır
when others => seven_segment <= "1111111";
end case;
when others => seven_segment <= "1111111";
end case;
```

```
end process;
end Behavioral;
```

Four Digit Display Testbench:

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

entity FourDigit_tb is
end FourDigit_tb;

architecture Behavioral of FourDigit_tb is
signal BIN_tb: std_logic_vector(13 downto 0);
signal clk_tb: std_logic;
signal seven_segment_tb: std_logic_vector (6 downto 0);
signal anode_tb: std_logic_vector (3 downto 0);

component FourDigit is
Port (
BIN: in std_logic_vector(13 downto 0);
clk: in std_logic;
seven_segment : out std_logic_vector (6 downto 0);
anode: out std_logic_vector (3 downto 0)
);
end component;

begin
UUT: FourDigit port map(
BIN => BIN_tb,
clk => clk_tb,
seven_segment => seven_segment_tb,
anode => anode_tb
);
```

```
process
begin
clk_tb <= '0';
wait for 10 ns;
clk_tb <= '1';
wait for 10 ns;
end process;

process
begin
BIN_tb <= "00000001111100";
wait for 50 ns;
BIN_tb <= "00101000011111";
end process;
end Behavioral;
```

Four Digit Display In Hexadecimal:

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

entity FourDigit is
Port (
BCDin: in std_logic_vector (31 downto 0);
clk: in std_logic;
seven_segment : out std_logic_vector (6 downto 0);
anode: out std_logic_vector (3 downto 0)

);
end FourDigit;
architecture Behavioral of FourDigit is
signal count: natural range 0 to 100000 :=0;
signal an_number: natural range 0 to 3 :=0;
```

```
signal countdiv: integer:=0;
signal tmp : std_logic := '0';
signal clock_out: std_logic;
begin
process(clk)
begin
if(rising_edge(clk)) then
count <=count+1;
if(count >= 100000) then
count <=0;
an_number <= an_number +1;
if (an_number>3) then
an_number <=0;
end if;
end if;
end if;
end process;
-- kodun k?sa hali için an_number case inin alt?nda BCDin(3 downto 0) yaz ve tek bir seven segment case'i yaz.
process(an_number)
begin
   case an_number is
   when 0 => anode <= "1110";
   when 1 => anode <= "1101";
   when 2 => anode <= "1011";
   when 3 => anode <= "0111";
   when others => anode <= "1111";
end case;
end process;

process(an_number)
```

```
begin
   case an_number is
   when 0=>
   case BCDin(3 downto 0) is
   when "0000" => seven_segment <= "0000001"; -- "0" olmazsa say?lar?n tersini
      when "0001" => seven_segment <= "1001111"; -- "1"
      when "0010" => seven_segment <= "0010010"; -- "2"
      when "0011" => seven_segment <= "0000110"; -- "3"
      when "0100" => seven_segment <= "1001100"; -- "4"
      when "0101" => seven_segment <= "0100100"; -- "5"
      when "0110" => seven_segment <= "0100000"; -- "6"
      when "0111" => seven_segment <= "0001111"; -- "7"
      when "1000" => seven_segment <= "0000000"; -- "8"
      when "1001" => seven_segment <= "0000100"; -- "9"
      when "1010" => seven_segment <= "0000010"; -- a
      when "1011" => seven_segment <= "1100000"; -- b
      when "1100" => seven_segment <= "0110001"; -- C
      when "1101" => seven_segment <= "1000010"; -- d
      when "1110" => seven_segment <= "0110000"; -- E
      when "1111" => seven_segment <= "0111000"; -- F
   when others => seven_segment <= "1111111";
   end case;
   when 1 =>
   case BCDin(7 downto 4) is
   when "0000" => seven_segment <= "0000001"; -- "0" olmazsa say?lar?n tersini
      when "0001" => seven_segment <= "1001111"; -- "1"
      when "0010" => seven_segment <= "0010010"; -- "2"
      when "0011" => seven_segment <= "0000110"; -- "3"
      when "0100" => seven_segment <= "1001100"; -- "4"
      when "0101" => seven_segment <= "0100100"; -- "5"
      when "0110" => seven_segment <= "0100000"; -- "6"
```

```
   when "0111" => seven_segment <= "0001111"; -- "7"
   when "1000" => seven_segment <= "0000000"; -- "8"
   when "1001" => seven_segment <= "0000100"; -- "9"
   when "1010" => seven_segment <= "0000010"; -- a
   when "1011" => seven_segment <= "1100000"; -- b
   when "1100" => seven_segment <= "0110001"; -- C
   when "1101" => seven_segment <= "1000010"; -- d
   when "1110" => seven_segment <= "0110000"; -- E
   when "1111" => seven_segment <= "0111000"; -- F
when others => seven_segment <= "1111111";
end case;
when 2 =>
case BCDin(11 downto 8) is
when "0000" => seven_segment <= "0000001"; -- "0" olmazsa say?lar?n tersini
   when "0001" => seven_segment <= "1001111"; -- "1"
   when "0010" => seven_segment <= "0010010"; -- "2"
   when "0011" => seven_segment <= "0000110"; -- "3"
   when "0100" => seven_segment <= "1001100"; -- "4"
   when "0101" => seven_segment <= "0100100"; -- "5"
   when "0110" => seven_segment <= "0100000"; -- "6"
   when "0111" => seven_segment <= "0001111"; -- "7"
   when "1000" => seven_segment <= "0000000"; -- "8"
   when "1001" => seven_segment <= "0000100"; -- "9"
   when "1010" => seven_segment <= "0000010"; -- a
   when "1011" => seven_segment <= "1100000"; -- b
   when "1100" => seven_segment <= "0110001"; -- C
   when "1101" => seven_segment <= "1000010"; -- d
   when "1110" => seven_segment <= "0110000"; -- E
   when "1111" => seven_segment <= "0111000"; -- F
when others => seven_segment <= "1111111";
end case;
```

```
   when 3 =>
   case BCDin(15 downto 12) is
   when "0000" => seven_segment <= "0000001"; -- "0" olmazsa say?lar?n tersini
      when "0001" => seven_segment <= "1001111"; -- "1"
      when "0010" => seven_segment <= "0010010"; -- "2"
      when "0011" => seven_segment <= "0000110"; -- "3"
      when "0100" => seven_segment <= "1001100"; -- "4"
      when "0101" => seven_segment <= "0100100"; -- "5"
      when "0110" => seven_segment <= "0100000"; -- "6"
      when "0111" => seven_segment <= "0001111"; -- "7"
      when "1000" => seven_segment <= "0000000"; -- "8"
      when "1001" => seven_segment <= "0000100"; -- "9"
      when "1010" => seven_segment <= "0000010"; -- a
      when "1011" => seven_segment <= "1100000"; -- b
      when "1100" => seven_segment <= "0110001"; -- C
      when "1101" => seven_segment <= "1000010"; -- d
      when "1110" => seven_segment <= "0110000"; -- E
      when "1111" => seven_segment <= "0111000"; -- F
   when others => seven_segment <= "1111111";
   end case;
   when others => seven_segment <= "1111111";
   end case;
end process;
end Behavioral;
```

Four Digit Display In Hexadecimal Testbench:

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

entity FourDigit_tb is
end FourDigit_tb;
```

```
architecture Behavioral of FourDigit_tb is
signal BIN_tb: std_logic_vector(13 downto 0);
signal clk_tb: std_logic;
signal seven_segment_tb: std_logic_vector (6 downto 0);
signal anode_tb: std_logic_vector (3 downto 0);

component FourDigit is
Port (
BIN: in std_logic_vector(13 downto 0);
clk: in std_logic;
seven_segment : out std_logic_vector (6 downto 0);
anode: out std_logic_vector (3 downto 0)

);
end component;
begin
UUT: FourDigit port map(
BIN => BIN_tb,
clk => clk_tb,
seven_segment => seven_segment_tb,
anode => anode_tb
);
process
begin
clk_tb <= '0';
wait for 10 ns;
clk_tb <= '1';
wait for 10 ns;
end process;
process
begin
```

```
BIN_tb <= "00000001111100";
wait for 50 ns;
BIN_tb <= "00101000011111";
end process;
end Behavioral;
```

CLK Divider:

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

entity ClkDiv is
port ( clk: in std_logic;
clkParameter: in integer;
clock_out: out std_logic);
end ClkDiv;

architecture bhv of ClkDiv is
signal count: integer:=0;
signal tmp : std_logic := '0';

begin
process(clk)
begin
if(clk'event and clk='1') then
   count <= count + 1;
   if (count = clkParameter) then
      tmp <= NOT tmp;
      count <= 0;
   end if;
end if;
clock_out <= tmp;
```

```
end process;


end bhv;
```

Shiftleft 2:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity shiftleft2 is
  generic(
    B: integer:=32;
    W: integer:=2
  );
        port (
                entry: in std_logic_vector(B-1 downto 0);
                exitt: out std_logic_vector(B-1 downto 0)
        );
end entity;
architecture behavioral of shiftleft2 is
signal temp: std_logic_vector(B-1 downto 0);
begin
        temp <= std_logic_vector(resize(unsigned(entry), B));
        exitt <= std_logic_vector(shift_left(signed(temp), 2));
end behavioral;
```