Assignment 5

Part-2 CPU

Due date: December 15, 2017

Group No: 5

Students:
Lalit Bhat: 01671833
Priya Parikh: 01679645

Objectives:

The objective of this LAB is to get hands on experience in learning to code in VHDL and to build a 4-bit CPU and test several instruction sets on the Cyclone VI FPGA kit.

List of components:

- Hardware used:

  Altera FPGA Cyclone IV board
  Power adapter
  Power supply

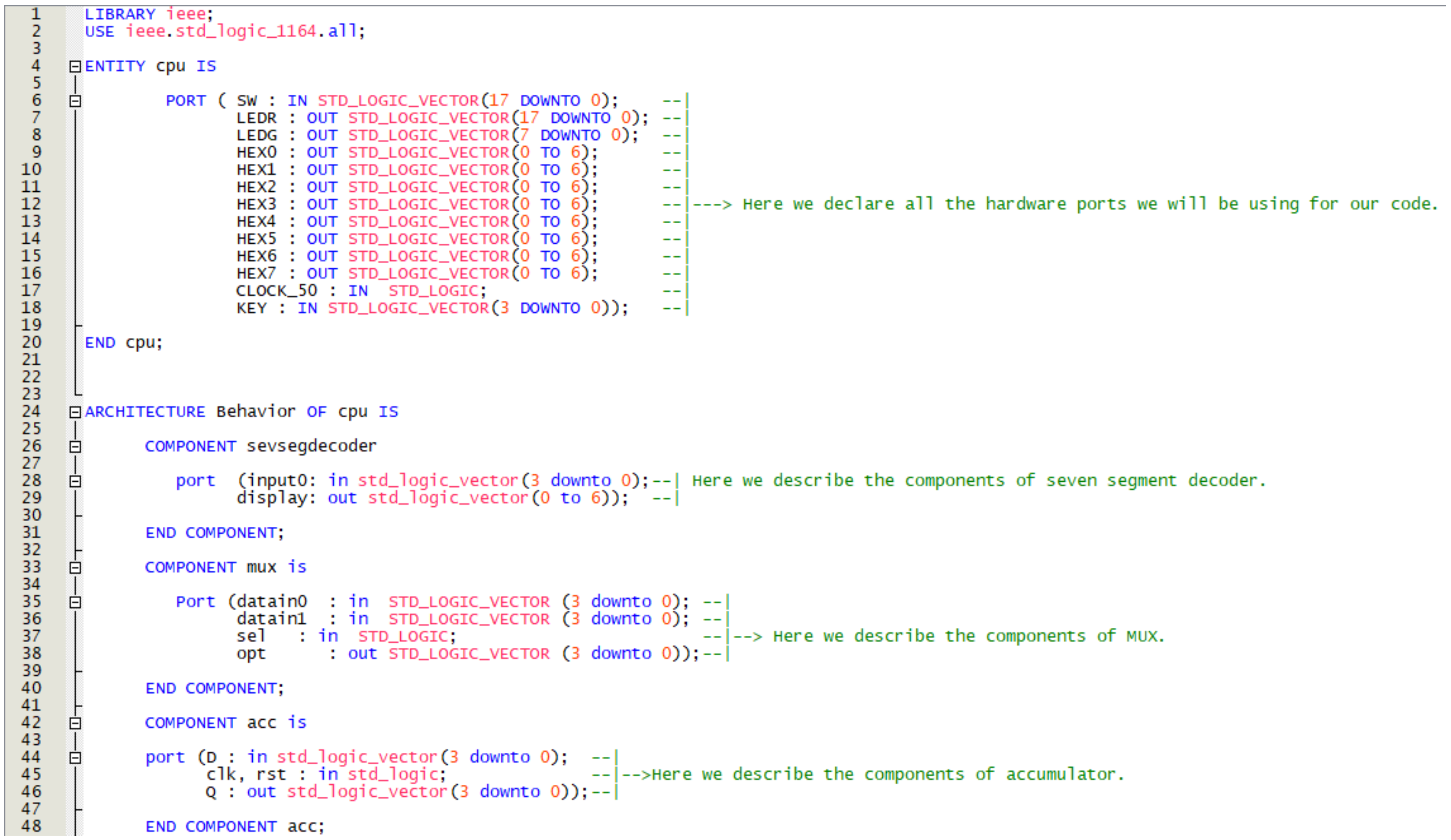
- Software used:
  Quartus Prime

Experimental Approach:

1. We first studied how to create a project in Quartus Prime.
2. After getting familiar with programming in VHDL we created individual projects of each components in the CPU.
3. We tested each of the individual projects on the cyclone IV kit
4. After designing the individual codes, we described the top level entity for the CPU.
5. We assigned the signals and described the port maps.
6. After building the final code we ran several simulations to find any bugs.
7. All 16 opcodes were tested and minor changes were made to the code to get the desired function.

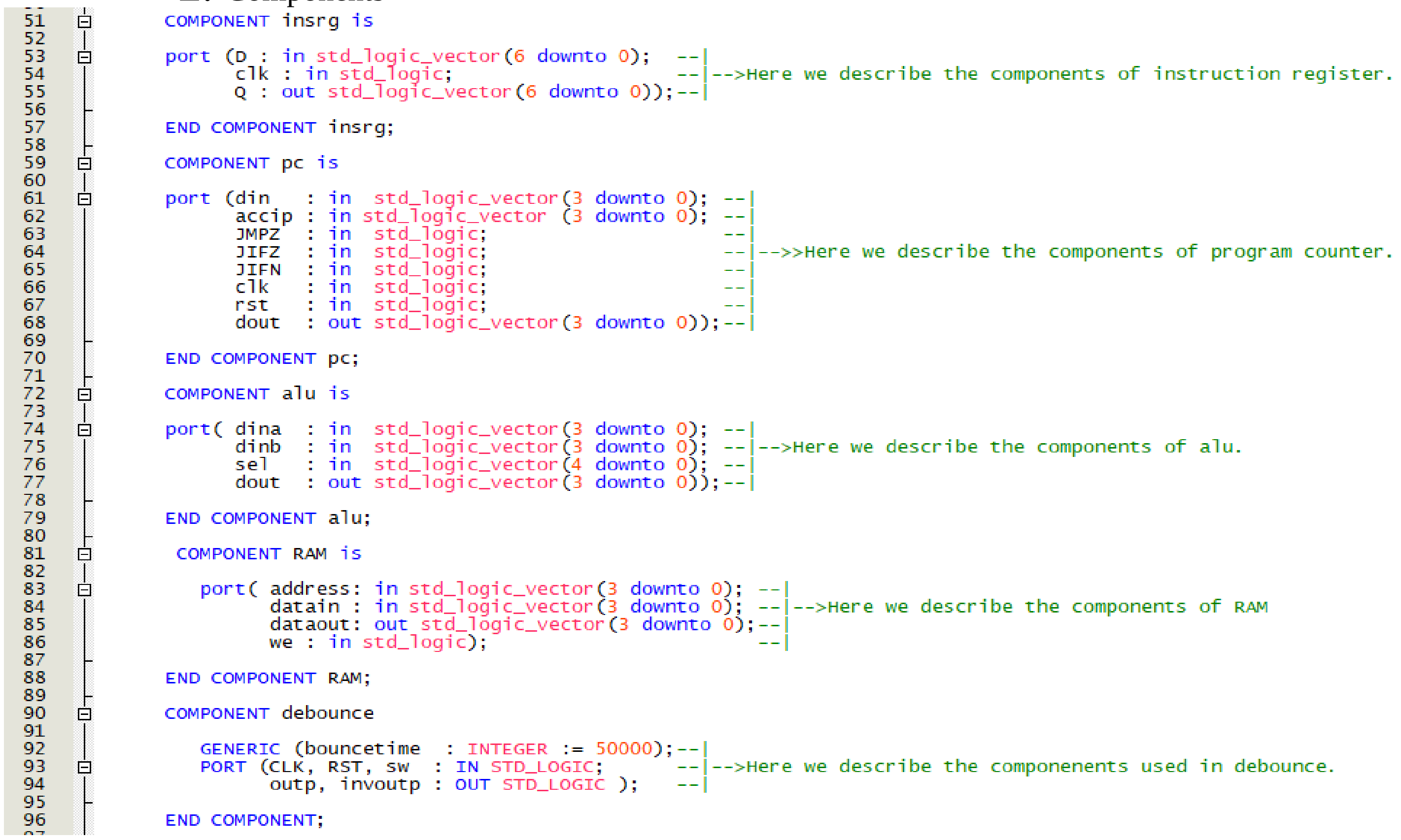
VHDL Code:

1. Top Level Entity.



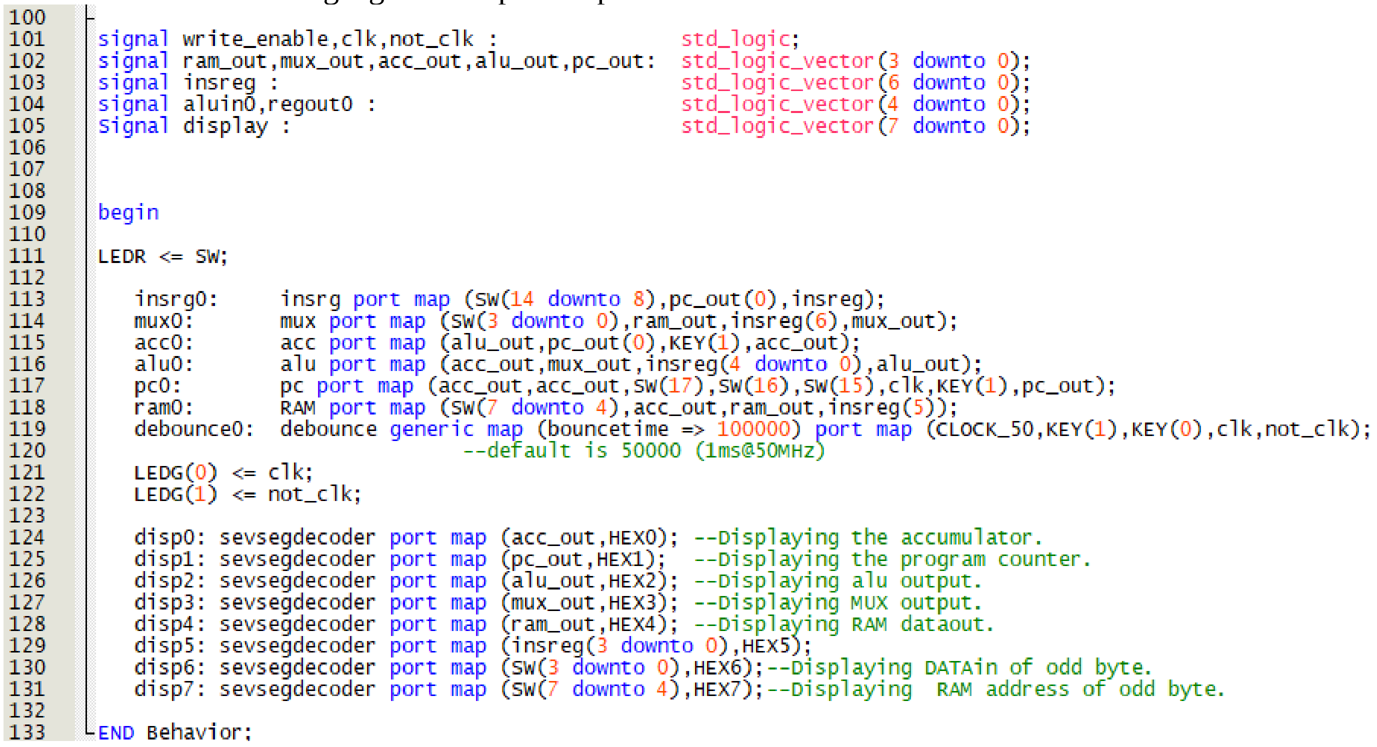
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY cpu IS

        PORT ( SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0);     --|
               LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0);  --|
               LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);   --|
               HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX1 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX2 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX3 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|---> Here we declare all the hardware ports we will be using for our code.
               HEX4 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX5 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX6 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               HEX7 : OUT STD_LOGIC_VECTOR(0 TO 6);       --|
               CLOCK_50 : IN  STD_LOGIC;                  --|
               KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0));    --|

END cpu;



ARCHITECTURE Behavior OF cpu IS

      COMPONENT sevsegdecoder

         port  (input0: in std_logic_vector(3 downto 0);--| Here we describe the components of seven segment decoder.
               display: out std_logic_vector(0 to 6));  --|

      END COMPONENT;

      COMPONENT mux is

         Port (datain0  : in  STD_LOGIC_VECTOR (3 downto 0); --|
               datain1  : in  STD_LOGIC_VECTOR (3 downto 0); --|
               sel    : in  STD_LOGIC;                       --|--> Here we describe the components of MUX.
               opt      : out STD_LOGIC_VECTOR (3 downto 0));--|

      END COMPONENT;

      COMPONENT acc is

      port (D : in std_logic_vector(3 downto 0);  --|
            clk, rst : in std_logic;              --|-->Here we describe the components of accumulator.
            Q : out std_logic_vector(3 downto 0));--|

      END COMPONENT acc;
```

2. Components



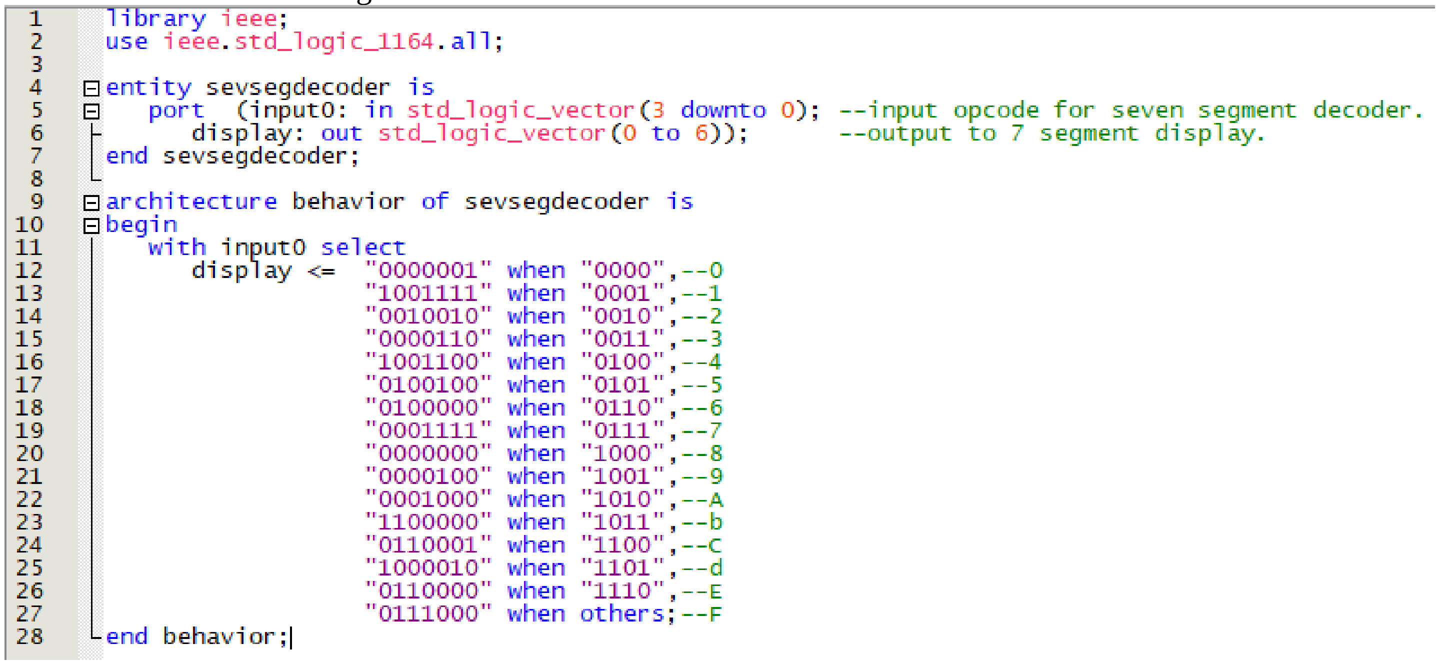
```
COMPONENT insrg is

port (D : in std_logic_vector(6 downto 0);   --|
      clk : in std_logic;                    --|-->Here we describe the components of instruction register.
      Q : out std_logic_vector(6 downto 0));--|

END COMPONENT insrg;

COMPONENT pc is

port (din   : in  std_logic_vector(3 downto 0); --|
      accip : in std_logic_vector (3 downto 0); --|
      JMPZ  : in  std_logic;                    --|
      JIFZ  : in  std_logic;                    --|-->>Here we describe the components of program counter.
      JIFN  : in  std_logic;                    --|
      clk   : in  std_logic;                    --|
      rst   : in  std_logic;                    --|
      dout  : out std_logic_vector(3 downto 0));--|

END COMPONENT pc;

COMPONENT alu is

port( dina  : in  std_logic_vector(3 downto 0); --|
      dinb  : in  std_logic_vector(3 downto 0); --|-->Here we describe the components of alu.
      sel   : in  std_logic_vector(4 downto 0); --|
      dout  : out std_logic_vector(3 downto 0));--|

END COMPONENT alu;

 COMPONENT RAM is

   port( address: in std_logic_vector(3 downto 0); --|
         datain : in std_logic_vector(3 downto 0); --|-->Here we describe the components of RAM
         dataout: out std_logic_vector(3 downto 0);--|
         we : in std_logic);                       --|

END COMPONENT RAM;

COMPONENT debounce

   GENERIC (bouncetime  : INTEGER := 50000);--|
   PORT (CLK, RST, sw  : IN STD_LOGIC;      --|-->Here we describe the componenents used in debounce.
         outp, invoutp : OUT STD_LOGIC );   --|

END COMPONENT;
```

3. Defining signals and port maps.



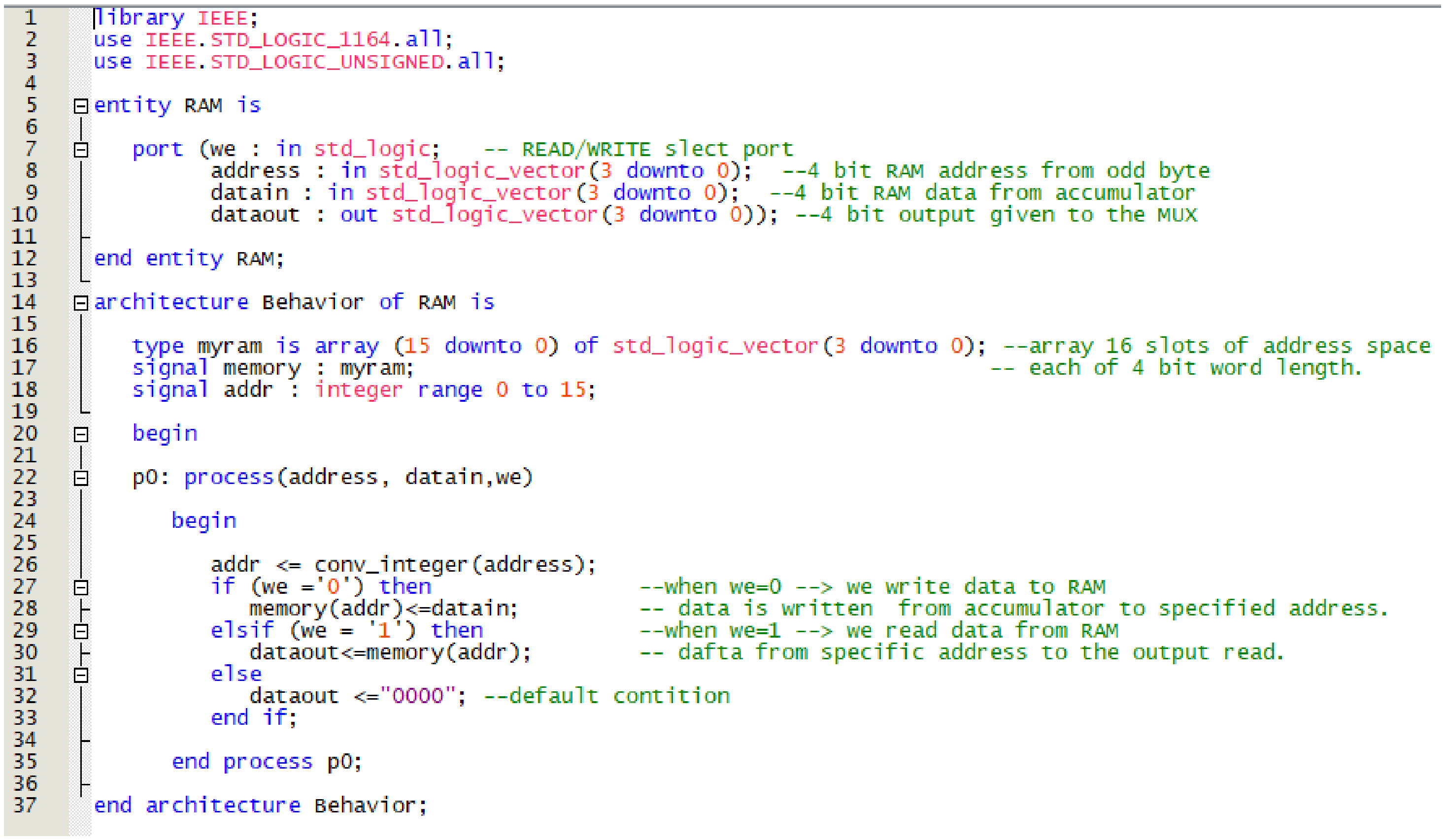
```
signal write_enable,clk,not_clk :              std_logic;
signal ram_out,mux_out,acc_out,alu_out,pc_out: std_logic_vector(3 downto 0);
signal insreg :                                std_logic_vector(6 downto 0);
signal aluin0,regout0 :                        std_logic_vector(4 downto 0);
signal display :                               std_logic_vector(7 downto 0);



begin

LEDR <= SW;

   insrg0:     insrg port map (SW(14 downto 8),pc_out(0),insreg);
   mux0:       mux port map (SW(3 downto 0),ram_out,insreg(6),mux_out);
   acc0:       acc port map (alu_out,pc_out(0),KEY(1),acc_out);
   alu0:       alu port map (acc_out,mux_out,insreg(4 downto 0),alu_out);
   pc0:        pc port map (acc_out,acc_out,SW(17),SW(16),SW(15),clk,KEY(1),pc_out);
   ram0:       RAM port map (SW(7 downto 4),acc_out,ram_out,insreg(5));
   debounce0:  debounce generic map (bouncetime => 100000) port map (CLOCK_50,KEY(1),KEY(0),clk,not_clk);
                              --default is 50000 (1ms@50MHz)
   LEDG(0) <= clk;
   LEDG(1) <= not_clk;

   disp0: sevsegdecoder port map (acc_out,HEX0); --Displaying the accumulator.
   disp1: sevsegdecoder port map (pc_out,HEX1);  --Displaying the program counter.
   disp2: sevsegdecoder port map (alu_out,HEX2); --Displaying alu output.
   disp3: sevsegdecoder port map (mux_out,HEX3); --Displaying MUX output.
   disp4: sevsegdecoder port map (ram_out,HEX4); --Displaying RAM dataout.
   disp5: sevsegdecoder port map (insreg(3 downto 0),HEX5);
   disp6: sevsegdecoder port map (SW(3 downto 0),HEX6);--Displaying DATAin of odd byte.
   disp7: sevsegdecoder port map (SW(7 downto 4),HEX7);--Displaying  RAM address of odd byte.

END Behavior;
```

4. Seven segment decoder.



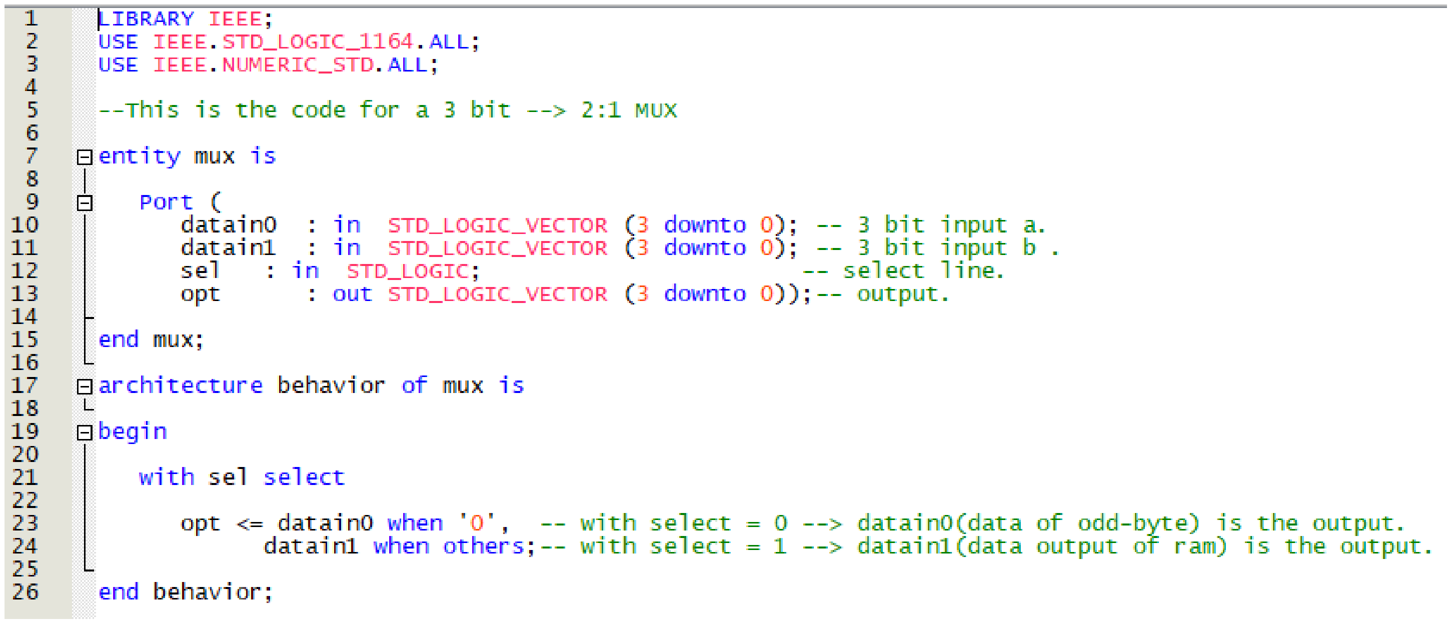
```
library ieee;
use ieee.std_logic_1164.all;

entity sevsegdecoder is
   port  (input0: in std_logic_vector(3 downto 0); --input opcode for seven segment decoder.
      display: out std_logic_vector(0 to 6));      --output to 7 segment display.
end sevsegdecoder;

architecture behavior of sevsegdecoder is
begin
   with input0 select
      display <=  "0000001" when "0000",--0
                  "1001111" when "0001",--1
                  "0010010" when "0010",--2
                  "0000110" when "0011",--3
                  "1001100" when "0100",--4
                  "0100100" when "0101",--5
                  "0100000" when "0110",--6
                  "0001111" when "0111",--7
                  "0000000" when "1000",--8
                  "0000100" when "1001",--9
                  "0001000" when "1010",--A
                  "1100000" when "1011",--b
                  "0110001" when "1100",--C
                  "1000010" when "1101",--d
                  "0110000" when "1110",--E
                  "0111000" when others;--F
end behavior;
```

5. RAM.



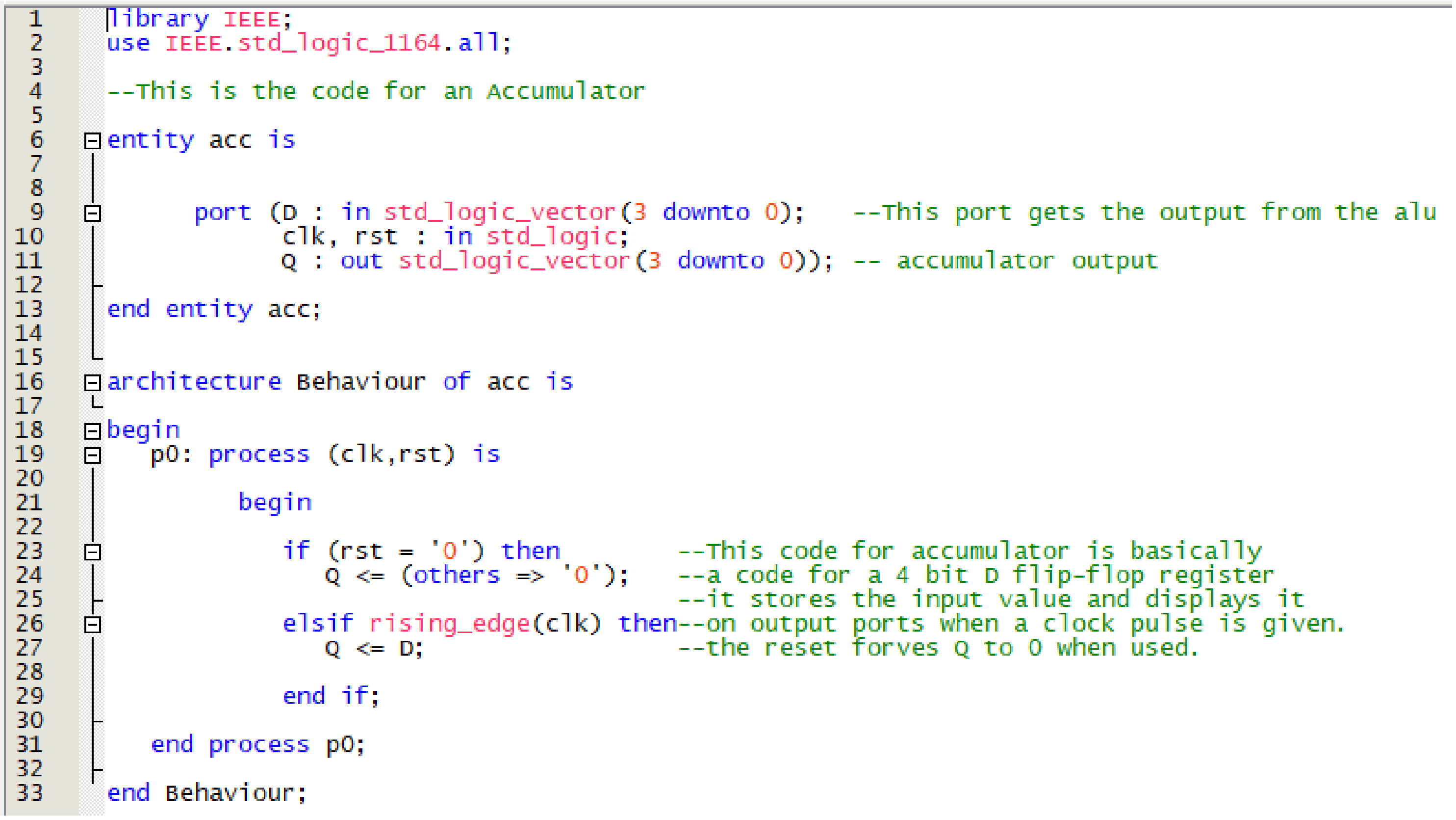
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity RAM is

   port (we : in std_logic;    -- READ/WRITE slect port
         address : in std_logic_vector(3 downto 0);  --4 bit RAM address from odd byte
         datain : in std_logic_vector(3 downto 0);   --4 bit RAM data from accumulator
         dataout : out std_logic_vector(3 downto 0)); --4 bit output given to the MUX

end entity RAM;

architecture Behavior of RAM is

   type myram is array (15 downto 0) of std_logic_vector(3 downto 0); --array 16 slots of address space
   signal memory : myram;                                             -- each of 4 bit word length.
   signal addr : integer range 0 to 15;

   begin

   p0: process(address, datain,we)

      begin

         addr <= conv_integer(address);
         if (we ='0') then                --when we=0 --> we write data to RAM
            memory(addr)<=datain;         -- data is written  from accumulator to specified address.
         elsif (we = '1') then            --when we=1 --> we read data from RAM
            dataout<=memory(addr);        -- dafta from specific address to the output read.
         else
            dataout <="0000"; --default contition
         end if;

      end process p0;

end architecture Behavior;
```

6. MUX.



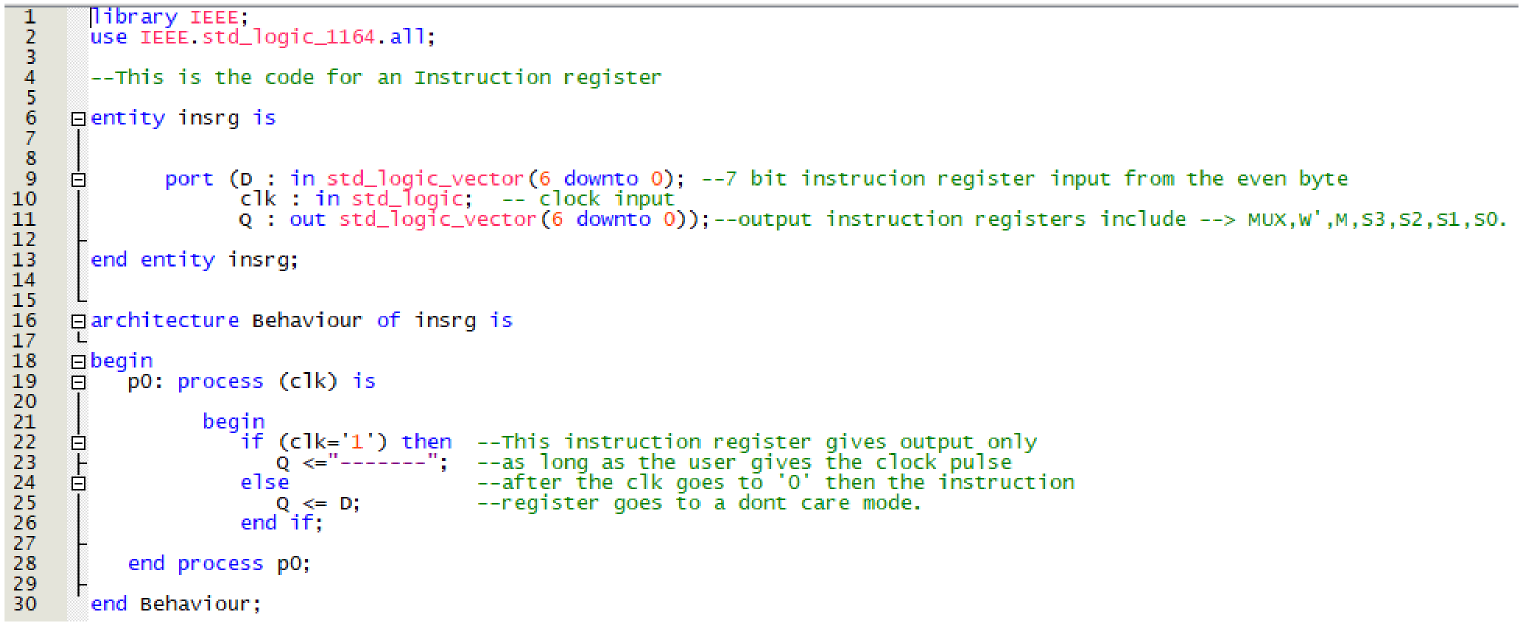
```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

--This is the code for a 3 bit --> 2:1 MUX

entity mux is

   Port (
      datain0  : in  STD_LOGIC_VECTOR (3 downto 0); -- 3 bit input a.
      datain1  : in  STD_LOGIC_VECTOR (3 downto 0); -- 3 bit input b .
      sel    : in  STD_LOGIC;                       -- select line.
      opt       : out STD_LOGIC_VECTOR (3 downto 0));-- output.

end mux;

architecture behavior of mux is

begin

   with sel select

      opt <= datain0 when '0',  -- with select = 0 --> datain0(data of odd-byte) is the output.
             datain1 when others;-- with select = 1 --> datain1(data output of ram) is the output.

end behavior;
```

7. Accumulator.



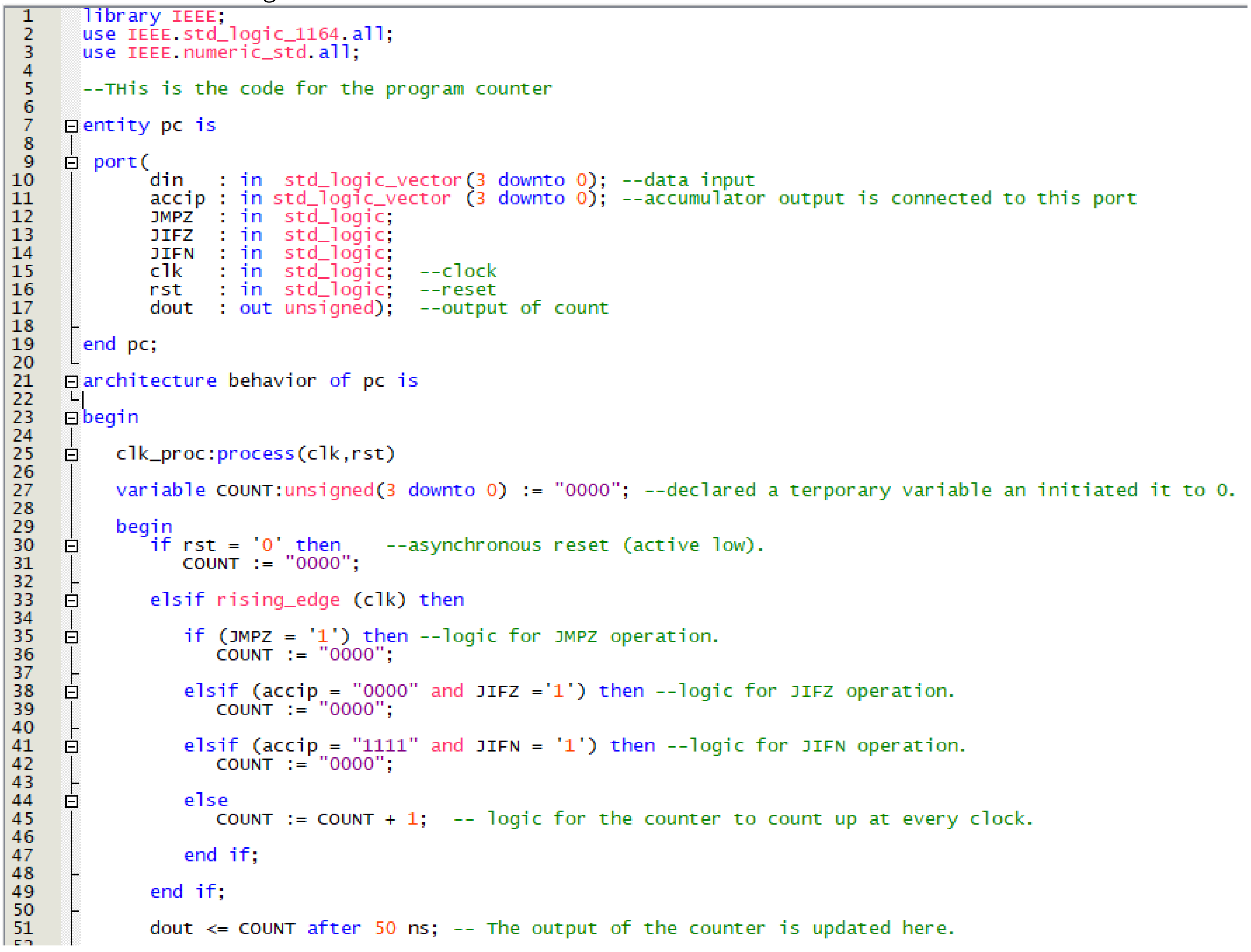
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

--This is the code for an Accumulator

entity acc is


        port (D : in std_logic_vector(3 downto 0);    --This port gets the output from the alu
              clk, rst : in std_logic;
              Q : out std_logic_vector(3 downto 0)); -- accumulator output

end entity acc;


architecture Behaviour of acc is

begin
   p0: process (clk,rst) is

         begin

            if (rst = '0') then         --This code for accumulator is basically
               Q <= (others => '0');    --a code for a 4 bit D flip-flop register
                                        --it stores the input value and displays it
            elsif rising_edge(clk) then--on output ports when a clock pulse is given.
               Q <= D;                  --the reset forves Q to 0 when used.

            end if;

   end process p0;

end Behaviour;
```

8. Instruction Register.

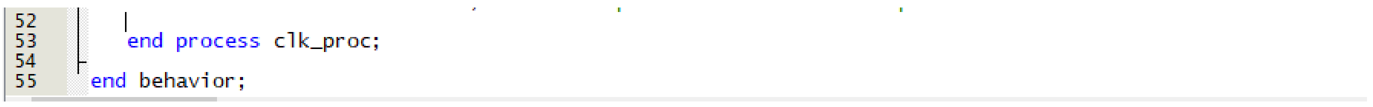


```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

--This is the code for an Instruction register

entity insrg is


      port (D : in std_logic_vector(6 downto 0); --7 bit instrucion register input from the even byte
            clk : in std_logic;  -- clock input
            Q : out std_logic_vector(6 downto 0));--output instruction registers include --> MUX,W',M,S3,S2,S1,S0.

end entity insrg;


architecture Behaviour of insrg is

begin
   p0: process (clk) is

         begin
            if (clk='1') then  --This instruction register gives output only
               Q <="-------";  --as long as the user gives the clock pulse
            else               --after the clk goes to '0' then the instruction
               Q <= D;         --register goes to a dont care mode.
            end if;

   end process p0;

end Behaviour;
```

9. Program Counter.

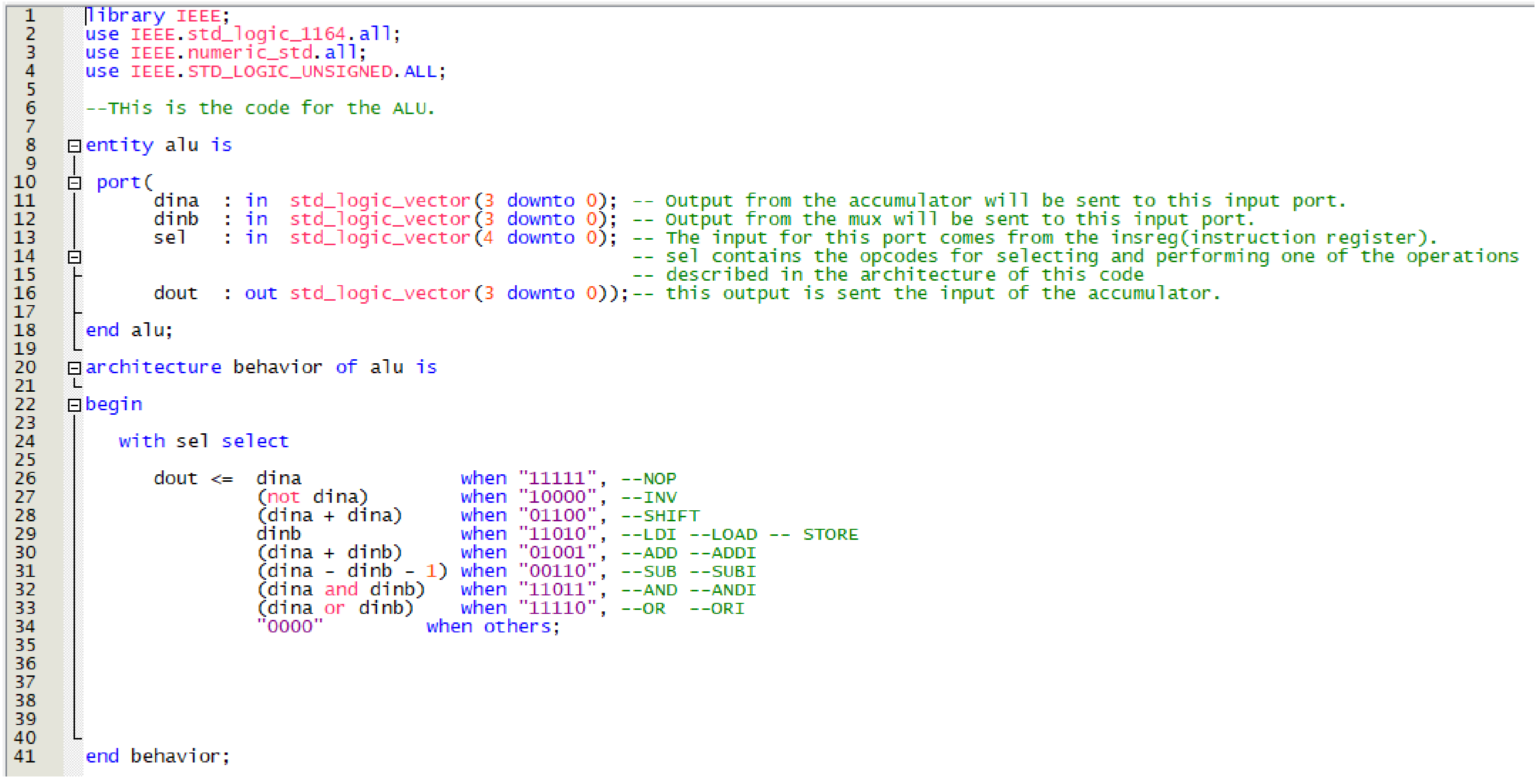


```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

--THis is the code for the program counter

entity pc is

 port(
       din   : in  std_logic_vector(3 downto 0); --data input
       accip : in std_logic_vector (3 downto 0); --accumulator output is connected to this port
       JMPZ  : in  std_logic;
       JIFZ  : in  std_logic;
       JIFN  : in  std_logic;
       clk   : in  std_logic;   --clock
       rst   : in  std_logic;   --reset
       dout  : out unsigned);   --output of count

end pc;

architecture behavior of pc is

begin

   clk_proc:process(clk,rst)

   variable COUNT:unsigned(3 downto 0) := "0000"; --declared a terporary variable an initiated it to 0.

   begin
      if rst = '0' then    --asynchronous reset (active low).
         COUNT := "0000";

      elsif rising_edge (clk) then

         if (JMPZ = '1') then --logic for JMPZ operation.
            COUNT := "0000";

         elsif (accip = "0000" and JIFZ ='1') then --logic for JIFZ operation.
            COUNT := "0000";

         elsif (accip = "1111" and JIFN = '1') then --logic for JIFN operation.
            COUNT := "0000";

         else
            COUNT := COUNT + 1;  -- logic for the counter to count up at every clock.

         end if;

      end if;

      dout <= COUNT after 50 ns; -- The output of the counter is updated here.
```



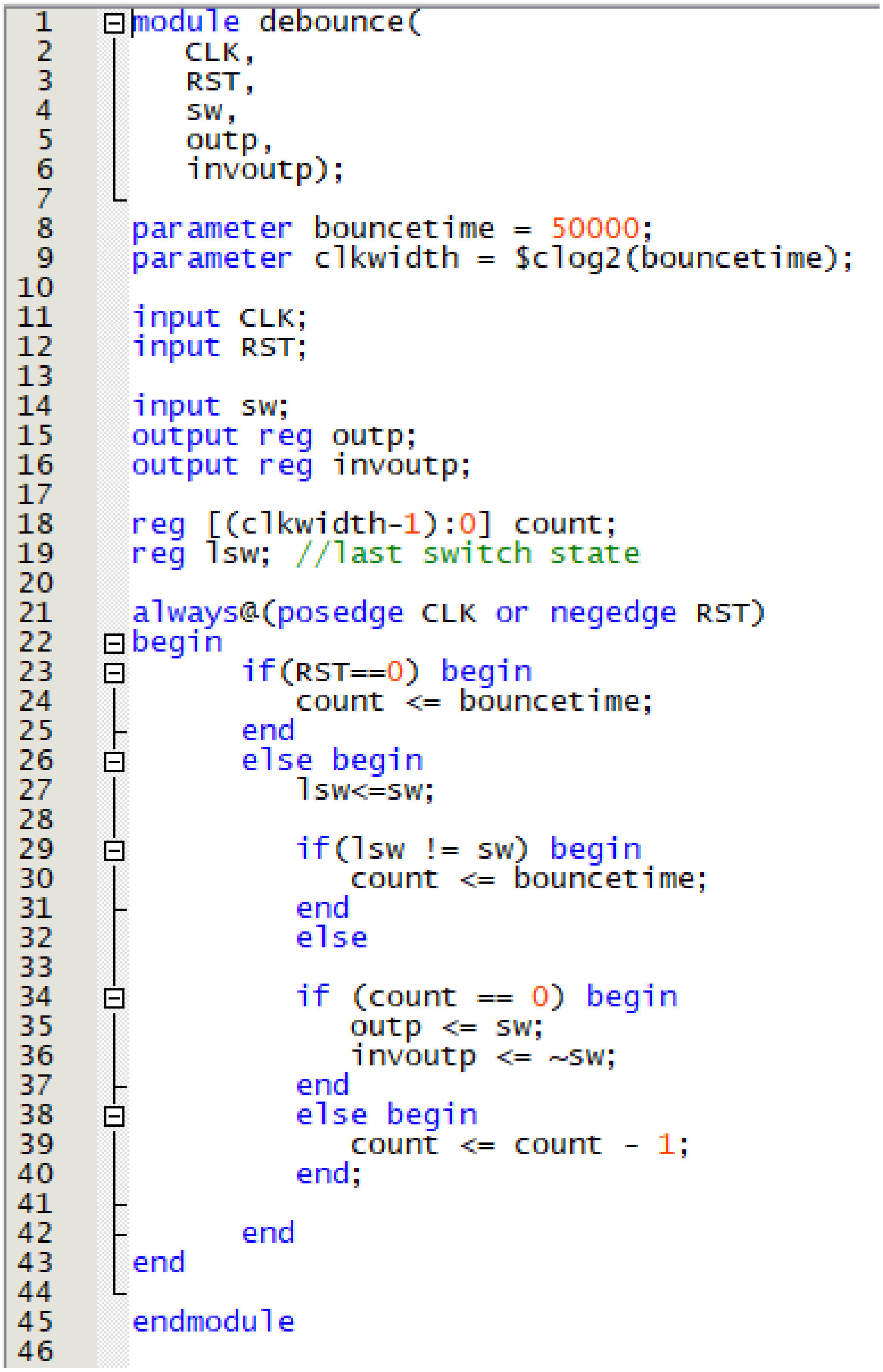
```vhdl

   end process clk_proc;

end behavior;
```

10. ALU



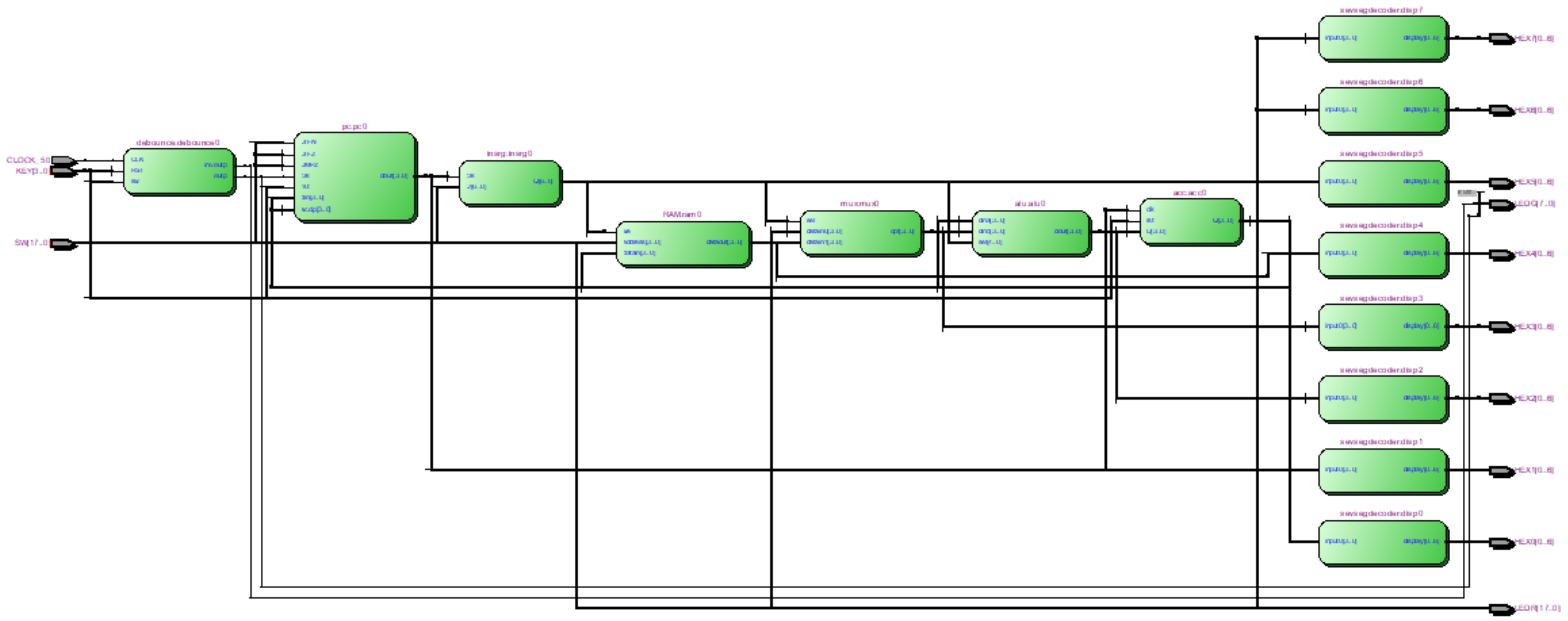
```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--THis is the code for the ALU.

entity alu is

 port(
      dina  : in  std_logic_vector(3 downto 0); -- Output from the accumulator will be sent to this input port.
      dinb  : in  std_logic_vector(3 downto 0); -- Output from the mux will be sent to this input port.
      sel   : in  std_logic_vector(4 downto 0); -- The input for this port comes from the insreg(instruction register).
                                                -- sel contains the opcodes for selecting and performing one of the operations
                                                -- described in the architecture of this code
      dout  : out std_logic_vector(3 downto 0));-- this output is sent the input of the accumulator.

end alu;

architecture behavior of alu is

begin

   with sel select

      dout <=  dina                 when "11111", --NOP
               (not dina)           when "10000", --INV
               (dina + dina)        when "01100", --SHIFT
               dinb                 when "11010", --LDI --LOAD -- STORE
               (dina + dinb)        when "01001", --ADD --ADDI
               (dina - dinb - 1)    when "00110", --SUB --SUBI
               (dina and dinb)      when "11011", --AND --ANDI
               (dina or dinb)       when "11110", --OR  --ORI
               "0000"            when others;




end behavior;
```

11. De-bounce.



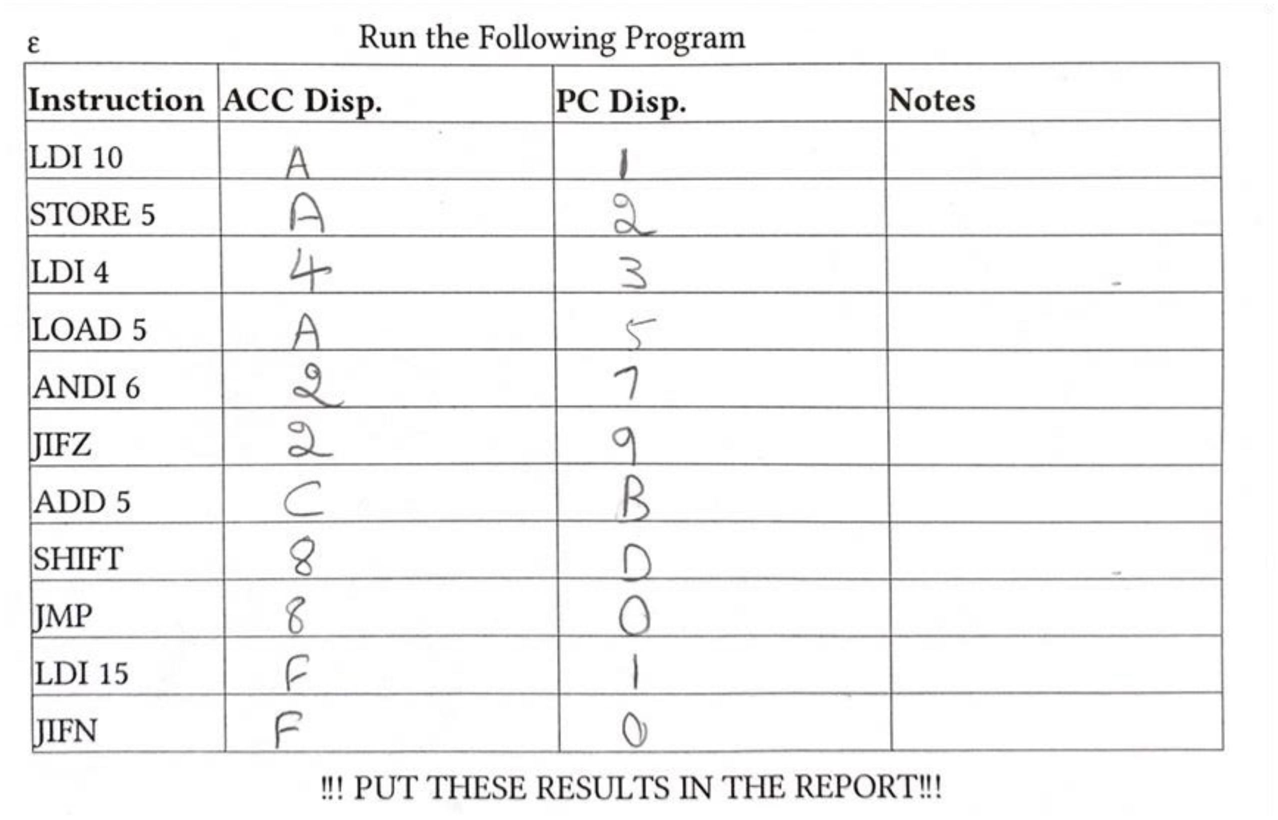
```
module debounce(
    CLK,
    RST,
    sw,
    outp,
    invoutp);

parameter bouncetime = 50000;
parameter clkwidth = $clog2(bouncetime);

input CLK;
input RST;

input sw;
output reg outp;
output reg invoutp;

reg [(clkwidth-1):0] count;
reg lsw; //last switch state

always@(posedge CLK or negedge RST)
begin
        if(RST==0) begin
            count <= bouncetime;
        end
        else begin
            lsw<=sw;

            if(lsw != sw) begin
                count <= bouncetime;
            end
            else

            if (count == 0) begin
                outp <= sw;
                invoutp <= ~sw;
            end
            else begin
                count <= count - 1;
            end;

        end
end

endmodule
```

RTL Viewer:


CLOCK_50
KEY[3..0]
SW[17..0]
debounce:debounce0
pc:pc0
insreg:insreg0
RAM:ram0
mux:mux0
alu:alu0
acc:acc0
sevsegdecoder:disp7
sevsegdecoder:disp6
sevsegdecoder:disp5
sevsegdecoder:disp4
sevsegdecoder:disp3
sevsegdecoder:disp2
sevsegdecoder:disp1
sevsegdecoder:disp0
HEX7[0..6]
HEX6[0..6]
HEX5[0..6]
LEDG[7..0]
HEX4[0..6]
HEX3[0..6]
HEX2[0..6]
HEX1[0..6]
HEX0[0..6]
LEDR[17..0]

Results:

The TA asked us to execute several RTL instructions in series and to note our observations of the clock and the program counter.
We were able to perform each of these instructions successfully.



ε

Run the Following Program

Instruction	ACC Disp.	PC Disp.	Notes
LDI 10	A	1	
STORE 5	A	2	
LDI 4	4	3	
LOAD 5	A	5	
ANDI 6	2	7	
JIFZ	2	9	
ADD 5	C	B	
SHIFT	8	D	
JMP	8	0	
LDI 15	F	1	
JIFN	F	0	

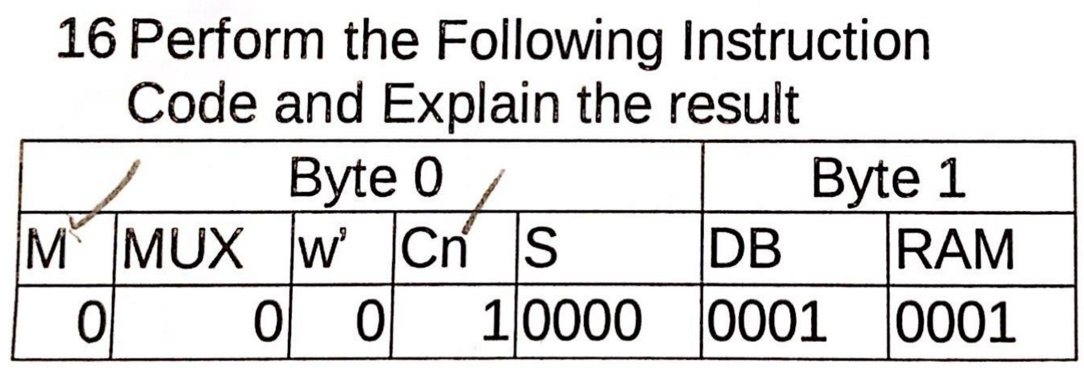
!!! PUT THESE RESULTS IN THE REPORT!!!

Conclusion:

We successfully designed and simulated VHDL code for the CPU using Quartus Prime.

FTQs:

1. **Lalit Bhat**



16 Perform the Following Instruction Code and Explain the result

Byte 0					Byte 1	
M	MUX	w’	Cn	S	DB	RAM
0	0	0	1	0000	0001	0001

ANSWER: After storing the value “0001” in DB and RAM, I entered the opcode for S, M, MUX, W’.
The Cn in my code is default logic 1 because I have used only logic high outputs.
The output according to the ALU designed in my code is ‘0’. Because I included the opcodes of only 16 instructions that were originally specified in the LAB 4 of CPU hardware.
So here the default condition in my code stores a ‘0’ in my accumulator.
If I were to program the above opcode in my ALU then the accumulator output will always display “0001” i.e. it acts as a NOP op-code.

2. **Priya Parekh**

Byte 0					Byte 1	
M	MUX	W’	Cn	S	DB	RAM
1	0	0	0	0001	0001	0000

Ans:

M	1	Shows mode control input high
MUX	0	Indicate RAM is not used
W’	0	write operation
Cn	0	Carry input is don’t care for the given operation
S	0001	Instruction execution (AB)’
DB	0001	data
RAM	0000	Register Location / address