```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_signed.all;
--------------------------- Declaring the entity microprocessor
--------------------------
entity mips is
port ( clk,reset: in std_logic;                      -- clock and reset inputs
 alu_load       : out std_logic;                     -- ALU operation control output
signal
 flag_status    : out std_logic;                     -- output flag status of ALU
 alu_operation  : out std_logic_vector(2 downto 0);  -- output ALU selector
 external_data  : in  std_logic_vector(7 downto 0);  -- 8 bit input data from input
port
 output_port    : out std_logic_vector(7 downto 0);  -- 8 bit output data to output
port
 a_register     : out std_logic_vector(7 downto 0);  -- A register(8bit)
 b_register     : out std_logic_vector(7 downto 0);  -- B register(8bit)
 op_code        : out std_logic_vector(2 downto 0);  -- opcode output
 pc_out         : out std_logic_vector(4 downto 0);  -- program counter output
 IR             : out std_logic_vector(7 downto 0);  -- instruction register output
 alu_result     : out std_logic_vector(7 downto 0)); -- accumlator output
end mips;

architecture Behavioral of mips is
---------------------------Signal Declaration for the  components of
Microprocess-----------
 signal pc_cur: std_logic_vector(4 downto 0);         -- 4 bit Program counter Signal
 signal jump_address: std_logic_vector(4 downto 0);   -- 4 bit jump Address Signal
 signal pcjump: std_logic;                            -- jump operation Signal
 signal instr: std_logic_vector(7 downto 0);          -- 8 bit instruction register
Signal
 signal opcode: std_logic_vector(7 downto 5);         -- opcode signal
 ------- Control Block Signal Declarations------------
 signal mem_read1,mem_write,mem_read2,inc_address2: std_logic;
 signal alu_op,acc_bus,pc_inc,lda_IR,lda_address,beq,ex_data,jal,lda_output: std_logi
 ------- Control Block Signal Declarations---------
 signal data_b: std_logic_vector(7 downto 0);           -- Data Register b signal
 signal result: std_logic_vector(7 downto 0);           -- Accumulator signal
 signal data_a: std_logic_vector(7 downto 0);           -- Data Register b signal
 signal flag: std_logic;                                -- flag signal of ALU
 ---------------------------------Sequential Circuit for Program
Counter------------------------------
begin
process(clk,reset)
begin
 if(reset='1') then                  -- check if active reset is set
```

```
  pc_cur <= "00000";                -- if active reset is set, reset the program
counter to zero
 elsif(rising_edge(clk)) then       -- when rising edge of clock is detected
  if(pc_inc='1') then               -- check if pc_inc is set
  pc_cur <= pc_cur + 1;             -- if pc_inc is set, the program counter is
incremented by 1
  elsif(pc_cur = "11111") then       -- check if the program counter is equal 1111
  pc_cur <= "00000";                 -- if the program counter is equal 1111, then it
will be reset
  elsif(pcjump = '1') then          -- check if pcjump is set
  pc_cur <=jump_address;           -- if pcjump is set, the program counter will be
changed to jump address value
  end if;
 end if;
 end process;
 ------------------------------------    Branch and jump operation
-----------------------------
process(flag,beq,jal)
begin
if(flag = '1' and beq = '1') then    -- check if flag and branch operation is set
pcjump <='1';                        -- if the previous coniditon was true, then
pcjump will be set
elsif(flag = '1' and jal = '1')then  -- check if flag and jump operation is set
pcjump <='1';                        -- if the previous coniditon was true, then
pcjump will be set
else
pcjump <='0';                        -- if all the conition was false in this
process, then pcjump will be reset to zerO
end if;
end process;
 ------------------------------------    output operation
-----------------------------
process(clk,reset)
begin
if(reset=  '1') then
output_port <= x"00";
elsif(rising_edge(clk)) then
if(lda_output = '1') then
output_port <=result;
end if;
end if;
end process;
-------------------------------- Mapping of Program Memory Design
---------------------------------------------
Instruction_Memory: entity work.program_memory
   port map
   (clk => clk,
    reset => reset,
    load_IR => lda_IR,
    address_load => lda_address,
    pc => pc_cur,
    instruction => instr);
-------------------------------- Mapping of Control Block Design
---------------------------------------------
control: entity work.control_block
   port map
   (reset => reset,
    clk => clk,
    inc_address=> inc_address2,
    opcode => instr(7 downto 5),
    ex_data => ex_data,
    Load_address => lda_address,
    load_output=>lda_output,
    branch_eq => beq,
    load_IR =>lda_IR,
    alu_op=> alu_op,
    acc_bus=> acc_bus,
    pc_inc=> pc_inc,
    jal=> jal,
    mem_read1 => mem_read1,
    mem_read2 => mem_read2,
    mem_write => mem_write);
-------------------------------- Mapping of Data Memory Design
---------------------------------------------
Data_memory: entity work.Data_Memory
     port map
    (clk => clk,
     reset => reset,
     inc_address => inc_address2,
     mem_address_Areg => instr(1 downto 0),
     mem_address_Breg => instr(4 downto 2),
     input_data => external_data,
     acc_bus=> acc_bus,
     load_ext=> ex_data,
     data_in => result,
     mem_write_en => mem_write,
     mem_read1 => mem_read1,
     mem_read2 => mem_read2,
     data_out1 => data_a,
     data_out2 => data_b);
-------------------------------- Mapping of ALU Design
---------------------------------------------
```

```
alu1: entity work.ALU
     port map
    (alu_op => alu_op,
     acc =>result,
     clk => clk,
     reset => reset,
     a_reg => data_a,
     b_reg => data_b,
     alu_control => instr(4 downto 2),
     flag => flag);
------------------------- Evalating the outputs of MIPS to observe them in testbench
simulation ---------------
   alu_load     <= alu_op;
   a_register   <= data_a;
   b_register   <= data_b;
   alu_result   <= result;
   pc_out       <= pc_cur;
   op_code      <=instr(7 downto 5);
   IR           <=instr;
   flag_status  <=flag;
   alu_operation<=instr(4 downto 2);
   jump_address <=instr(4 downto 0);

end Behavioral;
```