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

entity Control is
  port(
    rst : in std_logic;
    clk : in std_logic;

    -- Outputs
    ALUSrcA                                 : out std_logic;
    ALUSrcB                                 : out std_logic_vector( 1 downto 0 );
    ALUOp                                   : out std_logic_vector( 2 downto 0 );
    RegWrite                                : out std_logic;
    RegDst                                  : out std_logic_vector( 1 downto 0 );
    PCSource                                : out std_logic_vector( 1 downto 0 );
    PCLoad                                  : out std_logic;
    IorD                                    : out std_logic;
    MemRead                                 : out std_logic;
    MemWrite                                : out std_logic;
    MemtoReg                                : out std_logic_vector( 1 downto 0 );
    IRWrite                                 : out std_logic;
    Extend                                  : out std_logic;
    ReadSrc                                 : out std_logic;
    PCWriteCond                             : out std_logic_vector( 2 downto 0 );

    -- Inputs
    OpCode                                  : in std_logic_vector( 5 downto 0 );
    BranchType                              : in std_logic;
    funct                                   : in std_logic_vector( 5 downto 0 ));
end Control;


architecture bhv of Control is

  type state_type is(S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, s13, S14, S15,
s16);

  signal state, next_state : state_type;
  signal PCwrite           : std_logic;

begin  -- bhv

  process (clk, rst)
  begin
    -- Reset to S0
    if (rst = '1') then
      state <= S0;
      -- Otherwise go to the next cycle
    elsif (clk = '1' and clk'event) then
      state <= next_state;
    end if;
  end process;

-- state definitions
  process (rst, OpCode, BranchType, state, funct)
  begin
    -- Default values of the control signals
    ALUSrcA     <= '0';
    ALUSrcB     <= "00";
    ALUOp       <= "000";
    PCSource    <= "00";
    RegWrite    <= '0';
    RegDst      <= "00";
```

```
PCWrite     <= '0';
PCWriteCond <= "100";
IorD        <= '0';
MemRead     <= '0';
MemWrite    <= '0';
MemtoReg    <= "00";
IRWrite     <= '0';
Extend                                  <= '1';
ReadSrc                                 <= '0';

case state is

  -- Instruction Fetch
  when S0 =>
    ALUSrcA  <= '0';
    ALUSrcB  <= "01";
    ALUOp    <= "000";
    PCSource <= "00";
    PCWrite  <= '1';
    IorD     <= '0';
    MemRead  <= '1';
    IRWrite  <= '1';

    next_state <= S1;

    -- Instruction Decode / Register Fetch
  when S1 =>
    ALUSrcA <= '0';
    ALUSrcB <= "11";
    ALUOp   <= "000";

    case OpCode is
      when "100011" =>              -- Load
        next_state <= S2;
      when "001000" =>              -- Addi
        next_state <= S2;
      when "101011" =>              -- Store
        next_state <= S2;
      when "000100" =>              -- BEQ
        next_state <= S8;
      when "000010" =>              -- Jump
        next_state <= S9;
      when "001101" =>              -- Ori
        next_state <= S10;
      when "000111" =>
        next_state <= S12;                              -- Bgtz
      when "000101" =>
        next_state <= S13;                                 -- Bne
      when "001110" =>
        next_state <= S14;                 -- xori
      when "000001" =>
        next_state <= S15;                 -- bltzal
      when others   =>              -- R-Type
         case funct is
           when "001000" =>         -- JR
             next_state <= S16;
           when others =>
             next_state <= S6;
           end case;
      end case;

    -- Memory Address Computation
  when S2 =>
    ALUSrcA <= '1';
```

```
  ALUSrcB <= "10";
  ALUOp   <= "000";

  case OpCode is
    when "100011" =>     -- Memory access, Load
      next_state <= S3;
    when "101011" =>     -- Memory access, Store
      next_state <= S5;
    when "001000" =>     -- Addi
      next_state <= S11;

    when others   =>
  end case;

  -- Memory Read Access
when S3 =>
  IorD    <= '1';
  MemRead <= '1';

  next_state <= S4;

  -- Memory Read Completion
when S4 =>
  RegDst   <= "00";
  RegWrite <= '1';
  MemtoReg <= "01";

  next_state <= S0;

  -- Memory Write Access
when S5 =>
  IorD     <= '1';
  MemWrite <= '1';

  next_state <= S0;

  -- R-type Execution
when S6 =>
  ALUSrcA <= '1';
  ALUSrcB <= "00";
  ALUOp   <= "010";

  next_state <= S7;

  -- R-type Completion
when S7 =>
  RegWrite        <= '1';
  RegDst          <= "01";
  MemtoReg        <= "00";

  next_state      <= S0;

  -- BEQ Completion
when S8 =>
  ALUSrcA         <= '1';
  ALUSrcB         <= "00";
  ALUOp           <= "001";
  PCSource        <= "01";
  PCWriteCond     <= "011";

  next_state      <= S0;

  -- Jump Completion
when S9 =>
```

```
        PCSource        <= "10";
        PCWrite         <= '1';

        next_state      <= S0;


          -- Ori Start
      when S10 =>
        ALUSrcA <='1';
        ALUSrcB <="10";
        ALUOP   <="101";
        Extend  <= '0';

        next_state      <= S11;

-- ori,xori,addi Completion
      when S11 =>
                        RegDst <="00";
                        Extend  <= '0';
                        RegWrite <='1';
                        MemtoReg <="00";

                        next_state      <= S0;

-- bgtz Completion
          when S12 =>
                        ALUSrcA <= '1';
                        ALUSrcB <= "00";
                        PCWriteCond <= "000";
                        PCSource <= "01";

                        next_state <= S0;

-- bne Completion
     when S13 =>
                        ALUSrcA <= '1';
                        ALUSrcB <= "00";
                        ALUOP <= "001";
                        PCWriteCond <= "010";
                        PCSource <= "01";

                        next_state <= S0;

-- xori Start
      when S14 =>
                        ALUSrcA <= '1';
                        ALUSrcB <= "10";
                        ALUOP <= "100";
                        Extend <= '0';

                        next_state      <= S11;

-- bltzal Completion
      when S15 =>
                        ALUSrcA                         <= '1';
                        ALUSrcB                         <= "00";
                        ReadSrc                         <= '1';
                        PCWriteCond                             <= "001";
                        PCSource                                <= "01";
                        RegDst                          <= "10";
                        MemtoReg                                <= "10";
                        RegWrite                                <= '1';

                        next_state <= S0;
```

```
-- Jr completion
      when S16 =>
                          PCSource <= "11";
                          PCWrite <= '1';

                          next_state      <= S0;

      when others =>
    end case;
  end process;

-- Generate PCLoad control signal
  PCLoad <= PCWrite or BranchType;

end bhv;
```