```
 -- 8 to 1 mux with output
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
 entity Mux_8to1  is
    port(
      sel   : in  std_logic_vector(2 downto 0);
      d_in1 : in  std_logic;
      d_in2 : in  std_logic;
      d_in3 : in  std_logic;
      d_in4 : in  std_logic;
      d_in5 : in  std_logic;
      d_in6 : in  std_logic;
      d_in7 : in  std_logic;
      d_in8 : in  std_logic;
      d_out : out std_logic);
  end Mux_8to1;

 architecture bhv of Mux_8to1 is
begin
  with sel select
    d_out <=
    d_in1 when "000",
    d_in2 when "001",
    d_in3 when "010",
    d_in4 when "011",
    d_in5 when "100",
    d_in6 when "101",
    d_in7 when "110",
    d_in8 when others;
end bhv;

-- 2 to 1 mux with N-bit output
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity Mux_2to1_xN is
  generic(
    WIDTH :     integer := 32);
  port(
    sel   : in  std_logic;
    d_in1 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_in2 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end Mux_2to1_xN;

architecture bhv of Mux_2to1_xN is
begin
  with sel select
    d_out <=
    d_in1 when '0',
    d_in2 when others;
end bhv;

-- 4 to 1 mux with N-bit output
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;

entity Mux_4to1_xN is
  generic(
```

```
    WIDTH :     integer := 32);
  port(
    sel   : in  std_logic_vector(1 downto 0);
    d_in1 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_in2 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_in3 : in  std_logic_vector((WIDTH - 1)downto 0);
    d_in4 : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end Mux_4to1_xN;

architecture bhv of Mux_4to1_xN is
begin
  with sel select
    d_out <=
    d_in1 when "00",
    d_in2 when "01",
    d_in3 when "10",
    d_in4 when others;
end bhv;

-- instruction register
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity Reg_with_load is
  generic(
    WIDTH :     integer := 32);
  port(
    rst   : in  std_logic;
    clk   : in  std_logic;
    d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
    load  : in  std_logic;
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end Reg_with_load;

architecture bhv of Reg_with_load is
begin
  process(clk, rst)
  begin
    if(rst = '1') then
      d_out   <= (others => '0');
    elsif(clk'event and clk = '1') then
      if(load = '1') then
        d_out <= d_in;
      end if;
    end if;
  end process;
end bhv;

-- 32 entry register file
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;

entity Register_File is
  port(
    rst       : in  std_logic;
    clk       : in  std_logic;
    RegWrite  : in  std_logic;
    ReadReg1  : in  std_logic_vector(4 downto 0);
    ReadReg2  : in  std_logic_vector(4 downto 0);
    WriteReg  : in  std_logic_vector(4 downto 0);
```

```
    WriteData : in  std_logic_vector(31 downto 0);
    ReadData1 : out std_logic_vector(31 downto 0);
    ReadData2 : out std_logic_vector(31 downto 0));
end Register_File;

architecture bhv of Register_File is
  type Reg32Array is array (0 to 31) of std_logic_vector (31 downto 0);
  signal Registers : Reg32Array;

begin
  process(clk, rst, ReadReg1, ReadReg2)
  begin
    if(rst = '1') then
      for i in 0 to 31 loop
        Registers(i) <= (others => '0');
      end loop;
    else
      if(clk'event and clk = '1') then
        if(RegWrite = '1') then
          Registers(conv_integer(WriteReg)) <= WriteData;
        end if;
      end if;
    end if;
  end process;

  ReadData1 <= Registers(conv_integer(ReadReg1));
  ReadData2 <= Registers(conv_integer(ReadReg2));
end bhv;

-- 1 entry register file, for storing temporary output for another clock cycle
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity RegisterN is
  generic(
    WIDTH :     integer := 32);
  port(
    rst   : in  std_logic;
    clk   : in  std_logic;
    d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end RegisterN;

architecture bhv of RegisterN is
begin
  process(rst, clk)
  begin
    if(rst = '1') then
      d_out <= (others => '0');
    elsif(clk'event and clk = '1') then
      d_out <= d_in;
    end if;
  end process;
end bhv;

-- ALU
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity ALU is
  port(
    ctrl  : in  std_logic_vector(2 downto 0);
```

```
    d_in1 : in  std_logic_vector(31 downto 0);
    d_in2 : in  std_logic_vector(31 downto 0);
    shamt : in  std_logic_vector(4 downto 0);
    d_out : out std_logic_vector(31 downto 0);
    Zero  : out std_logic;
    Positive  : out std_logic);
end ALU;

architecture bhv of ALU is
begin
  process(ctrl, d_in1, d_in2)
    variable d_aux : std_logic_vector(31 downto 0);
  begin

    case ctrl is
      when "011"  =>    d_aux := d_in2;                   -- sll
                        for i in 1 to conv_integer(shamt) loop
                                d_aux := d_aux(30 downto 0) & '0';
                        end loop;

      when "000"  => d_aux   := d_in1 and d_in2;  -- and
      when "001"  => d_aux   := d_in1 or d_in2;   -- or
      when "010"  => d_aux   := d_in1 + d_in2;    -- add
      when "110"  => d_aux   := d_in1 - d_in2;    -- sub
      when "100"  => d_aux   := d_in1 xor d_in2;  -- xor
      when "111"  => if(d_in1 < d_in2) then       -- slt
                       d_aux := (0 => '1', others => '0');
--"00000000000000000000000000000001";
                     else
                       d_aux := (others => '0');
                     end if;
      when others => d_aux   := (others => 'X');
    end case;
    if (d_aux = "00000000000000000000000000000000") then
      Zero <= '1';
    else
      Zero <= '0';
    end if;

    if (conv_integer(d_aux) > 0) then
      Positive <= '1';
    else
      Positive <= '0';
    end if;

    d_out  <= d_aux;
  end process;
end bhv;

-- sign extender, 16 bits to 32 bits
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity Sign_extend is
  port(
    d_in  : in  std_logic_vector(15 downto 0);
    d_out : out std_logic_vector(31 downto 0));
end Sign_extend;

architecture bhv of Sign_extend is
begin
  process(d_in)
  begin
```

```
    for i in 31 downto 16 loop
      d_out(i)         <= d_in(15);
    end loop;
    d_out(15 downto 0) <= d_in(15 downto 0);
  end process;
end bhv;

-- zero extender, 16 bits to 32 bits
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity Zero_extend is
  port(
    d_in  : in  std_logic_vector(15 downto 0);
    d_out : out std_logic_vector(31 downto 0));
end Zero_extend;

architecture bhv of Zero_extend is
begin
  process(d_in)
  begin
    for i in 31 downto 16 loop
      d_out(i)         <= '0';
    end loop;
    d_out(15 downto 0) <= d_in(15 downto 0);
  end process;
end bhv;


-- shift left 2
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
entity Shift_left_2 is
  generic(
    WIDTH :     integer := 32);
  port(
    d_in  : in  std_logic_vector((WIDTH - 1) downto 0);
    d_out : out std_logic_vector((WIDTH - 1) downto 0));
end Shift_left_2;

architecture bhv of Shift_left_2 is
begin
  d_out((WIDTH - 1) downto 2) <= d_in((WIDTH - 3) downto 0);
  d_out(1 downto 0)           <= "00";
end bhv;


-- Memory, not coded by me
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_textio.all;
library std;
use std.textio.all;

entity Memory is
  port(
    rst   : in  std_logic;
    clk   : in  std_logic;
    rd    : in  std_logic;
```

```
    wr    : in  std_logic;
    addr  : in  std_logic_vector (31 downto 0);
    d_in  : in  std_logic_vector (31 downto 0);
    d_out : out std_logic_vector (31 downto 0)
    );
end Memory;

architecture bhv of Memory is

  type MEM_TYPE is array (0 to 63) of std_logic_vector (31 downto 0);
  signal MEMORY : MEM_TYPE;

begin
  process( clk, rst )
    file fname    : text is in "mem.txt";  -- memory content
    variable L    : line;
    variable word : std_logic_vector(31 downto 0);
    variable i    : integer := 0;
  begin
    if ( rst'event and rst = '1') then
      while (not(endfile(fname)) and (i < 64)) loop
        readline(fname, L);
        hread(L, word);
        MEMORY(i) <= word;
        i := i + 1;
      end loop;
    elsif (clk'event and clk = '1') then
      if (wr = '1') then
        MEMORY(conv_integer("00" & addr (7 downto 2))) <= d_in;
      end if;
    end if;
  end process;

      d_out <= MEMORY(conv_integer("00" & addr (7 downto 2))) when (rd = '1') else
               (others => 'Z');

    end bhv;
```