```vhdl
-- May Trinh, Payton Fisher, Danny Woodard
-- Logic Design
-- Mini Processor Project

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity main is
   port(

      upd, exe: in std_logic; -- 2 clocks
      instruction : in std_logic_vector(15 downto 0);
      out1: out std_logic_vector(6 downto 0);
      out2: out std_logic_vector (6 downto 0)
   );

end entity;

-- architecture for main entity

architecture rtl of main is

-- component to decode the instruction
component Decoder is
   port(
      instruction : in std_logic_vector (15 downto 0);
   -- upd         : in std_logic; -- update destination register
   -- exe         : in std_logic; -- execute the operation
   -- exe_out     : out std_logic;
   -- upd_out     : out std_logic;
      desReg      : out std_logic;
      chosenInstr : out std_logic_vector(3 downto 0);
      isSub       : out std_logic;
      isConst     : out std_logic
   );

end component;

component registerEnabler is

   port
   (
      desReg      : in std_logic;
      upd      : in std_logic;
      enableR0  : out std_logic;
      enableR1  : out std_logic
   );
end component;


-- register entity
component reg is
   port(
      rin: in std_logic_vector(7 downto 0);--input register
      enable: in std_logic;
      rout : out std_logic_vector (7 downto 0)
   );
end component;


-- multiplexor for register
component regMux is

   port
   (
      r0: in std_logic_vector (7 downto 0);
      r1: in std_logic_vector (7 downto 0);
```

```vhdl
         s: in std_logic;
         y: out std_logic_vector (7 downto 0)
      );

   end component;

   -- multiplexor for operator

   component mux is
      port (
         instr   : in std_logic_vector (3 downto 0);
         addorsub  : in std_logic_vector (7 downto 0);
         sub :in std_logic_vector (7 downto 0);
         isConst: in std_logic;
         addConst: in std_logic_vector (7 downto 0);
         subConst: in std_logic_vector (7 downto 0);
         xand    : in std_logic_vector (7 downto 0);
         mov     : in std_logic_vector (7 downto 0);
         neg      : in std_logic_vector (7 downto 0);
         Eor      : in std_logic_vector (7 downto 0);
         shl     : in std_logic_vector  (7 downto 0);
         shr     : in std_logic_vector (7 downto 0);
         to_reg : out std_logic_vector (7 downto 0)
      );
   end component;

   component mov is
      port(
         num: in std_logic_vector (7 downto 0);
         rd : out std_logic_vector (7 downto 0)
      );
   end component;

   component display is
      port(
         rs: in std_logic_vector ( 7 downto 0);
         a_seg: out std_logic_vector (6 downto 0);
         b_seg : out std_logic_vector (6 downto 0)
      );
   end component;

   component add is
      port (a,b: in std_logic_vector(7 downto 0);
         isSub : in std_logic;
         sum: out std_logic_vector (7 downto 0)
      );
   end component;

   component subtract is
   port (a,b: in std_logic_vector(7 downto 0);
         isSub : in std_logic;
         sum: out std_logic_vector (7 downto 0)
      );
   end component;

   component getConst is
      port (
         a: in std_logic_vector(2 downto 0);
         isConst : in std_logic;
         new_val: out std_logic_vector(7 downto 0)
      );
   end component;

   component andReg is
      port (
         r0,r1: in std_logic_vector (7 downto 0);
         rd: out std_logic_vector (7 downto 0)
      );
```

```vhdl
end component;



-- neg entity - complement Rs, stores in Rd
component neg is
   port (
      rs: in std_logic_vector (7 downto 0);
      rd: out std_logic_vector (7 downto 0)
   );
end component;

-- eor entity-- xor Rs and Rd, stores in Rd
component eor is
   port(
      r0, r1 : in std_logic_vector(7 downto 0);
      rd: out std_logic_vector(7 downto 0)
   );
end component;


------------------------------------------------------------------------
-- lsl entity - left shift Rs by 1 bit, stores in Rd
component lsl is
   port (
      rs: in std_logic_vector (7 downto 0);
      rd: out std_logic_vector (7 downto 0)
   );
end component;


component lsr is
   port (
      rs: in std_logic_vector (7 downto 0);
      rd: out std_logic_vector (7 downto 0)
   );
end component;


signal v: std_logic_vector(7 downto 0);
signal rs: std_logic_vector (0 downto 0);
signal reg0, reg1, reg3: std_logic_vector(7 downto 0);
signal rs1, rs2, rs3: std_logic_vector(7 downto 0);

-- for multiplexor instruction
signal addSubAns, constAddAns, subAns,subConstAns, andAns, movAns, negAns, eorAns, shlAns,
shrAns, finalAns : std_logic_vector (7 downto 0);
signal chosenRegv, displayRegv: std_logic_vector ( 7 downto 0);
signal r0_new, r1_new :std_logic_vector (7 downto 0);

--decoder's needs
signal exe_outM, upd_outM, wRegM, enableAL_M, enableBL_M, isSubM, isConstM: std_logic;
signal chosenInstrM: std_logic_vector (3 downto 0);

--for register
signal r0_outM, r1_outM, tempOut: std_logic_vector(7 downto 0);
signal a1,b1 : std_logic;

begin


      --D: Decoder port map(instruction, (not upd), (not exe), exe_outM, upd_outM, wRegM,
chosenInstrM, isSubM, isConstM);
      D: Decoder port map (instruction, wRegM, chosenInstrM, isSubM, isConstM);
      --regE: registerEnabler port map(wRegM, upd_outM, enableAL_M, enableBL_M);

      -- operator move
      a: mov port map (instruction(7 downto 0), movAns);
```

```vhdl

        wReg1: regMux port map(r0_outM, r1_outM, instruction(6), reg0);
        wReg2: regMux port map(r0_outM, r1_outM, instruction(3), reg1);

        addReg: add port map (reg0, reg1, isSubM, addSubAns);
        sub: subtract port map (reg0, reg1, isSubM, subAns);

        const: getConst port map(instruction(8 downto 6), isConstM, r0_new);

        -- add/subtract constant
        addConst: add port map (r0_new, reg1, isSubM, constAddAns);
        subConst: subtract port map (reg1,r0_new, isSubM, subConstAns);


        wReg3: regMux port map(r0_outM, r1_outM, instruction(0), reg3);
        -- eor operator
        xorReg: eor port map (reg1, reg3, eorAns);

        -- and operator
        andRegister: andReg port map (reg1, reg3, andAns);

        -- multiplexor to decide the register for operation
        wRegPass: regMux port map(r0_outM, r1_outM, instruction(3), chosenRegv);

        -- neg operator
        negate: neg port map (chosenRegv, negAns);

        -- shift_left operator
        shl: lsl port map (chosenRegv, shlAns);

        -- shift_right operator
        shr: lsr port map (chosenRegv, shrAns);

        -- answers passed through a multiplexor to determine the instruction
        multiplexor: mux port map (chosenInstrM, addSubAns, subAns, isConstM, constAddAns,
subConstAns, andAns, movAns, negAns, eorAns, shlAns, shrAns, finalAns);


        regE: registerEnabler port map(wRegM, upd, enableAL_M, enableBL_M);

        -- temp register to store the value and passes it to R0 or R1 when necessary
        temp: reg port map (finalAns, (not exe), tempOut);
        r0: reg port map (tempOut, enableAL_M, r0_outM);
        r1: reg port map (tempOut, enableBL_M, r1_outM);


        displayReg: regMux port map (r0_outM, r1_outM, wRegM, displayRegv);
        displ: display port map (displayRegv, out1, out2);

end;


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-----------------------------------------------------------------
entity Decoder is
   port(
      instruction : in std_logic_vector (15 downto 0);
   -- upd         : in std_logic; -- update destination register
   -- exe         : in std_logic; -- execute the operation
   -- exe_out     : out std_logic;
   -- upd_out     : out std_logic;
      desReg      : out std_logic;
      chosenInstr : out std_logic_vector(3 downto 0);
      isSub       : out std_logic;
      isConst     : out std_logic
   );

end entity;

architecture rtl of Decoder is

signal firstFour: std_logic_vector (3 downto 0);
signal reg: std_logic_vector (2 downto 0);
signal rd: std_logic;
signal instrC: std_logic_vector (3 downto 0);
signal updd, exed: std_logic;
begin

   firstFour <= instruction(15 downto 12);

   process(instruction) is
   begin

      if (firstFour = "0010") then -- move instruction
         reg <= instruction(10 downto 8);

      else
         reg <= instruction(2 downto 0);-- destination register
      end if;

      if(reg = "000") then
         rd <= '0'; -- destination register is R0
      else
         rd <= '1'; -- destination register is R1
      end if;
   end process;

   process(instruction) is
   begin

      --add instruction
      if(instruction (15 downto 9 ) = "0001100") then
         instrC <= "0000";
         isSub <= '0';
         isConst <= '0';

      -- add constant
      elsif (instruction (15 downto 9) = "0001110") then
         instrC <= "1001";
         isSub <= '0';
         isConst <= '1';

      -- and instruction
      elsif ((instruction (15 downto 6) = "0100000000")) then
         instrC <= "0001";

      -- mov instruction
      elsif((instruction (15 downto 11) = "00100")) then
         instrC <= "0010";

      -- negate instruction
      elsif((instruction (15 downto 6) = "0100001001")) then
         instrC <= "0011";

      -- xor instruction
      elsif((instruction (15 downto 6)= "0100000001")) then
         instrC <= "0100";

      -- shift left instruction
      elsif((instruction (15 downto 6) = "0000000001")) then
         instrC <= "0101";

      -- shift right instruction
```

```vhdl
        elsif((instruction (15 downto 6) = "0000100001")) then
           instrC <= "0110";

        -- subtract instruction
        elsif(instruction (15 downto 9) = "0001101") then
           isSub <= '1';
           instrC <= "0111";
           isConst <='0';

        -- subtract constant
        elsif(instruction (15 downto 9) = "0001111") then
           isSub <= '1';
           instrC <= "1000";
           isConst <= '1';
        end if;

     end process;

  -- process(upd, exe) is
  -- begin
  --
  --    if(upd = '0' and exe = '0') then -- both clock cannot be ON, -> set both to OFF
  --       updd <= '1';
  --       exed <= '1';
  --    else
  --       updd <= upd;
  --       exed <= exe;
  --    end if;
  -- end process;
  --
  -- exe_out <= exed;
  -- upd_out <= updd;
     desReg <= rd; -- destination register
     chosenInstr <= instrC; -- chosen instruction
  end rtl;


  -----------------------------------
  library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
  -------------------------------------------------------
  -- register entity
  entity reg is
     port(
        rin: in std_logic_vector(7 downto 0);--input register
        enable: in std_logic;
        rout : out std_logic_vector (7 downto 0)
     );
  end entity;


  architecture r of reg is
  signal rt: std_logic_vector (7 downto 0);
  begin
     process (enable) is
     begin

        if (enable = '1') then
           rout <= rin;
        end if;

     end process;
  end;

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

```

entity registerEnabler is

   port
   (
      desReg      : in std_logic;
      upd      : in std_logic;
      enableR0   : out std_logic;
      enableR1   : out std_logic
   );

end entity;

architecture rtl of registerEnabler is
 signal setA, setB: std_logic;
begin

   process(upd, desReg) is
   begin

   if(upd = '1') then -- clock is off when it's ONE
   --None of the registers are updated
      setA <= '0';
      setB <= '0';
   elsif (upd = '0') then -- clock is ON when it's ZERO
      -- if 0 -> R0 is updated, if 1-> R1 is updated
      if(desReg = '0') then -- update R0
         setA <= '1';
         setB <= '0';
      else -- Update R1
         setA <= '0';
         setB <= '1';
      end if;

   end if;
   end process;

   enableR0 <= setA;
   enableR1 <= setB;

-- if (upd ='0') then
--    if(desReg ='0') then
--       enableR0 <= '1';
--    else
--       enableR1 <= '1';
--    end if;
-- end if;
-- end process;

end rtl;


--------------------------------------------------------------
--Reg Multiplexor: Decides which register value is to be used

library ieee;
use ieee.std_logic_1164.all;

entity regMux is

   port
   (
      r0: in std_logic_vector (7 downto 0);
      r1: in std_logic_vector (7 downto 0);
      s: in std_logic;
      y: out std_logic_vector (7 downto 0)
   );

```

```vhdl
end entity;

architecture rtl of regMux is
signal currAns: std_logic_vector (7 downto 0);
begin

   process(r0,r1, s) is
   begin

      if (s = '0') then
         currAns <= r0;
      else
         currAns <= r1;

      end if;

   end process;
   y <= currAns;

end rtl;


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
---------------------------------------------------------------
-- multiplexer to select the instruction

entity mux is
   port (
      instr   : in std_logic_vector (3 downto 0);
      addorsub  : in std_logic_vector (7 downto 0);
      sub :in std_logic_vector (7 downto 0);
      isConst: in std_logic;
      addConst: in std_logic_vector (7 downto 0);
      subConst: in std_logic_vector (7 downto 0);
      xand    : in std_logic_vector (7 downto 0);
      mov     : in std_logic_vector (7 downto 0);
      neg      : in std_logic_vector (7 downto 0);
      Eor      : in std_logic_vector (7 downto 0);
      shl     : in std_logic_vector  (7 downto 0);
      shr     : in std_logic_vector (7 downto 0);
      to_reg : out std_logic_vector (7 downto 0)
   );
end entity;

architecture synth of mux is
 signal currAns: std_logic_vector (7 downto 0);

begin
   process (instr, isConst)
      begin
      if (instr = "0000") then -- add instruction
         currAns <= addorsub;

      elsif (instr ="1001") then --add const
         currAns <= addConst;

      elsif(instr = "0111") then  -- subtract instruction
         currAns <= sub;

      elsif (instr = "1000") then
         currAns <= subConst;-- subtract constant

      elsif (instr = "0001") then
         --and
         currAns <= xand;
      elsif(instr = "0010") then
```

```
            --mov
            currAns <= mov;
         elsif(instr = "0011") then
            --neg
            currAns <= neg;
         elsif(instr = "0100") then
            --or
            currAns <= Eor;
         elsif(instr = "0101") then
            --shl
            currAns <= shl;
         elsif(instr = "0110") then
            --shr
            currAns <= shr;

         end if;

      end process;

      to_reg <= currAns;

   end;


   ----------------------------------------------------------------
   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

   -- mov entity
   --Move 8 bit value to Rd
   entity mov is
      port(
         num: in std_logic_vector (7 downto 0);
         rd : out std_logic_vector (7 downto 0)
      );
   end entity;

   architecture m of mov is
   begin
      rd <= num;
   end;

   -----------------------------------------------------------------

   library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
   --------------------------------------------------------------------------
   -- out entity- display
   entity display is
      port (
         rs: in std_logic_vector ( 7 downto 0);
         a_seg: out std_logic_vector (6 downto 0);
         b_seg : out std_logic_vector (6 downto 0)
      );
   end entity;

   architecture d of display is
   signal a,b: std_logic_vector (3 downto 0);

   begin
         a <= rs(7 downto 4);
         b <= rs(3 downto 0);
      process(a)
         begin
            case a is
               when "0000" =>
```

```vhdl
                a_seg <= "0000001"; --0

            when "0001" =>
                a_seg <= "1001111"; --1

            when "0010" =>
                a_seg <= "0010010"; --2

            when "0011" =>
                a_seg <= "0000110"; --3

            when "0100" =>
                a_seg <= "1001100"; --4

            when "0101" =>
                a_seg <= "0100100"; --5

            when "0110" =>
                a_seg <= "0100000"; --6

            when "0111" =>
                a_seg <= "0001111"; --7

            when "1000" =>
                a_seg <= "0000000"; --8

            when "1001" =>
                a_seg <= "0000100"; --9

            when "1010" =>
                a_seg <= "0001000"; --A

            when "1011" =>
                a_seg <= "1100000"; --b

            when "1100" =>
                a_seg <= "0110001"; --C

            when "1101" =>
                a_seg <= "1000010"; --d

            when "1110" =>
                a_seg <= "0110000"; --E

            when "1111" =>
                a_seg <= "0111000"; --F

        end case;
    end process;
    process (b)
        begin
        case b is

            when "0000" =>
                b_seg <= "0000001"; --0

            when "0001" =>
                b_seg <= "1001111"; --1

            when "0010" =>
                b_seg <= "0010010"; --2

            when "0011" =>
                b_seg <= "0000110"; --3

            when "0100" =>
                b_seg <= "1001100"; --4

                when "0101" =>
                   b_seg <= "0100100"; --5

                when "0110" =>
                   b_seg <= "0100000"; --6

                when "0111" =>
                   b_seg <= "0001111"; --7

                when "1000" =>
                   b_seg <= "0000000"; --8

                when "1001" =>
                   b_seg <= "0000100"; --9

                when "1010" =>
                   b_seg <= "0001000"; --A

                when "1011" =>
                   b_seg <= "1100000"; --b

                when "1100" =>
                   b_seg <= "0110001"; --C

                when "1101" =>
                   b_seg <= "1000010"; --d

                when "1110" =>
                   b_seg <= "0110000"; --E

                when "1111" =>
                   b_seg <= "0111000"; --F

             end case;
          end process;

    end;

    ---------------------------------------------------------

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

    -- full adder for 1 bit register
    entity oneBitAdder is
          port (a, b, carry_in : in std_logic;
                 carry_out, sum : out std_logic);
    end entity;

    architecture rt1 of oneBitAdder is
    signal a1, a2, a3: std_logic;
    begin

       sum <= a xor b xor carry_in;
       carry_out <= (a and b) or (carry_in and a) or (carry_in and b);


    end rt1;



    --------------------------------------------------------------

    library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;

```

```vhdl
entity add is
   port (a,b: in std_logic_vector(7 downto 0);
      isSub : in std_logic;
      sum: out std_logic_vector (7 downto 0)
   );
end entity;
architecture a of add is

   signal c1, c2, c3, c4, c5, c6, c7, c8: std_logic;

   component oneBitAdder
      port (a, b, carry_in : in std_logic;
             carry_out, sum : out std_logic);
   end component;


begin

   add0: oneBitAdder port map(a(0), b(0), '0', c1,sum(0));
   add1: oneBitAdder port map(a(1), b(1), c1 ,c2, sum(1));
   add2: oneBitAdder port map(a(2), b(2), c2, c3, sum(2));
   add3: oneBitAdder port map(a(3), b(3), c3, c4, sum(3));
   add4: oneBitAdder port map(a(4), b(4), c4, c5, sum(4));
   add5: oneBitAdder port map(a(5), b(5), c5, c6 ,sum(5));
   add6: oneBitAdder port map(a(6), b(6), c6, c7 ,sum(6));
   add7: oneBitAdder port map(a(7), b(7), c7, c8 ,sum(7));

end;
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity subtract is
port (a,b: in std_logic_vector(7 downto 0);
      isSub : in std_logic;
      sum: out std_logic_vector (7 downto 0)
   );
end entity;
architecture s of subtract is

   signal c1, c2, c3, c4, c5, c6, c7, c8: std_logic;

   component oneBitAdder
      port (a, b, carry_in : in std_logic;
             carry_out, sum : out std_logic);
   end component;


begin

   add0: oneBitAdder port map(a(0), not b(0), '1', c1,sum(0));
   add1: oneBitAdder port map(a(1), not b(1), c1 ,c2, sum(1));
   add2: oneBitAdder port map(a(2), not b(2), c2, c3, sum(2));
   add3: oneBitAdder port map(a(3), not b(3), c3, c4, sum(3));
   add4: oneBitAdder port map(a(4), not b(4), c4, c5, sum(4));
   add5: oneBitAdder port map(a(5), not b(5), c5, c6 ,sum(5));
   add6: oneBitAdder port map(a(6), not b(6), c6, c7 ,sum(6));
   add7: oneBitAdder port map(a(7), not b(7), c7, c8 ,sum(7));

end;

--------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity getConst is
   port (
        a: in std_logic_vector(2 downto 0);
        isConst : in std_logic;
        new_val: out std_logic_vector(7 downto 0)
    );
end entity;

architecture gC of getConst is

begin
    process(a, isConst)
    begin
        if (isConst = '1') then
            new_val <= (7 downto 3 => '0') & a;
        end if;
    end process;
end;
-----------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
----------------------------------------------------------------------
-- and entity, and Rs and Rd, stores in Rd

entity andReg is
    port (
        r0,r1: in std_logic_vector (7 downto 0);
        rd: out std_logic_vector (7 downto 0)
    );
end entity;

architecture aR of andReg is
begin
    rd <= r0 and r1;

end;

-----------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
------------------------------------------------------------------------
-- neg entity - complement Rs, stores in Rd
entity neg is
    port (
        rs: in std_logic_vector (7 downto 0);
        rd: out std_logic_vector (7 downto 0)
    );
end entity;

architecture n of neg is
begin
    rd <= not rs;
end;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-------------------------------------------------------------------
-- eor entity-- xor Rs and Rd, stores in Rd
entity eor is
    port(
        r0, r1 : in std_logic_vector(7 downto 0);
        rd: out std_logic_vector(7 downto 0)
    );
```

```vhdl
end entity;

architecture e of eor is
begin
   rd <= r0 xor r1;
end;


--------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

---------------------------------------------------------------------------
-- lsl entity - left shift Rs by 1 bit, stores in Rd
entity lsl is
   port (
      rs: in std_logic_vector (7 downto 0);
      rd: out std_logic_vector (7 downto 0)
   );
end entity;

architecture l of lsl is
signal shl_temp : unsigned(7 downto 0);
begin

   shl_temp <= unsigned(rs);
   rd <= std_logic_vector (shift_left(shl_temp,1));
end;

--------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
---------------------------------------------------------------------------
-- lsr entity - right shift Rs by 1 bit, stores in Rd
entity lsr is
   port (
      rs: in std_logic_vector (7 downto 0);
      rd: out std_logic_vector (7 downto 0)
   );
end entity;

architecture r of lsr is

signal shr_temp : unsigned(7 downto 0);
begin

   shr_temp <= unsigned(rs);

   rd <= std_logic_vector(shift_right(shr_temp,1));
end;
```