```
--
-- TeC7 VHDL Source Code
--    Tokuyama kousen Educational Computer Ver.7
--
-- Copyright (C) 2002-2011 by
--                      Dept. of Computer Science and Electronic Engineering,
--                      Tokuyama College of Technology, JAPAN
--
--   上記著作権者は，Free Software Foundation によって公開されている GNU 一般公
-- 衆利用許諾契約書バージョン２に記述されている条件を満たす場合に限り，本ソース
-- コード(本ソースコードを改変したものを含む．以下同様)を使用・複製・改変・再配
-- 布することを無償で許諾する．
--
--   本ソースコードは＊全くの無保証＊で提供されるものである。上記著作権者および
-- 関連機関・個人は本ソースコードに関して，その適用可能性も含めて，いかなる保証
-- も行わない．また，本ソースコードの利用により直接的または間接的に生じたいかな
-- る損害に関しても，その責任を負わない．
--
--
-- TeC CPU VHDL Source Code
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

library work;

entity TEC_CPU is
  port ( P_CLK   : in  std_logic;                        -- Clock
         P_RESET : in  std_logic;                        -- Reset
         P_ADDR  : out std_logic_vector(7 downto 0);     -- ADDRESS BUS
         P_DIN   : in  std_logic_vector(7 downto 0);     -- DATA    BUS
         P_DOUT  : out std_logic_vector(7 downto 0);     -- DATA    BUS
         P_LI    : out std_logic;                        -- Instruction Fetch
         P_HL    : out std_logic;                        -- Halt Request
         P_ER    : out std_logic;                        -- Decode Error
         P_RW    : out std_logic;                        -- Read/Write
         P_MR    : out std_logic;                        -- Memory Request
         P_IR    : out std_logic;                        -- I/O Request
         P_INTR  : in  std_logic;                        -- Interrupt
         P_STOP  : in  std_logic;                        -- Stop

         P_WRITE : in  std_logic;                        -- Panel Write
         P_SEL   : in  std_logic_vector(2 downto 0);     -- Panel RotarySW Pos
         P_PND   : in  std_logic_vector(7 downto 0);     -- Panel Data
         P_C     : out std_logic;                        -- Carry   Flag
         P_S     : out std_logic;                        -- Sign    Flag
         P_Z     : out std_logic;                        -- Zero    Flag
         P_G0D   : out std_logic_vector(7 downto 0);     -- G0 out
         P_G1D   : out std_logic_vector(7 downto 0);     -- G1 out
         P_G2D   : out std_logic_vector(7 downto 0);     -- G2 out
         P_SPD   : out std_logic_vector(7 downto 0);     -- SP out
         P_PCD   : out std_logic_vector(7 downto 0);     -- PC out

         P_MODE  : in  std_logic                         -- DEMO MODE
       );
end TEC_CPU;

architecture RTL of TEC_CPU is

constant ALU_ZERO : std_logic_vector(3 downto 0) := "0000";
constant ALU_A    : std_logic_vector(3 downto 0) := "0001";
constant ALU_B    : std_logic_vector(3 downto 0) := "0010";
constant ALU_ADD  : std_logic_vector(3 downto 0) := "0011";
constant ALU_SUB  : std_logic_vector(3 downto 0) := "0100";
```



```
constant ALU_AND  : std_logic_vector(3 downto 0) := "0101";
constant ALU_OR   : std_logic_vector(3 downto 0) := "0110";
constant ALU_XOR  : std_logic_vector(3 downto 0) := "0111";
constant ALU_INC  : std_logic_vector(3 downto 0) := "1000";
constant ALU_DEC  : std_logic_vector(3 downto 0) := "1001";
constant ALU_ADC  : std_logic_vector(3 downto 0) := "1010";
constant ALU_OUTF : std_logic_vector(3 downto 0) := "1011";
constant ALU_INF  : std_logic_vector(3 downto 0) := "1100";
constant ALU_SFT  : std_logic_vector(3 downto 0) := "1101";

-- レジスタの指定
constant REG_SP   : std_logic_vector(1 downto 0) := "00";
constant REG_PC   : std_logic_vector(1 downto 0) := "01";
constant REG_Rd   : std_logic_vector(1 downto 0) := "10";
constant REG_Rx   : std_logic_vector(1 downto 0) := "11";

-- ジャンプの条件
constant JMP_NO   : std_logic_vector(2 downto 0) := "000";
constant JMP_ALL  : std_logic_vector(2 downto 0) := "001";
constant JMP_OP   : std_logic_vector(2 downto 0) := "010";
constant JMP_INT  : std_logic_vector(2 downto 0) := "011";
constant JMP_STP  : std_logic_vector(2 downto 0) := "100";
constant JMP_JP   : std_logic_vector(2 downto 0) := "101";
constant JMP_NJP  : std_logic_vector(2 downto 0) := "110";
constant JMP_DI   : std_logic_vector(2 downto 0) := "111";

-- レジスタファイル
signal I_G0  : std_logic_vector(7 downto 0);  -- G0
signal I_G1  : std_logic_vector(7 downto 0);  -- G1
signal I_G2  : std_logic_vector(7 downto 0);  -- G2
signal I_SP  : std_logic_vector(7 downto 0);  -- SP
signal I_PC  : std_logic_vector(7 downto 0);  -- PC

signal I_WRA : std_logic_vector(2 downto 0);  -- Write Address
signal I_RRA : std_logic_vector(2 downto 0);  -- Read  Address
signal I_WRD : std_logic_vector(7 downto 0);  -- Write Data
signal I_RRD : std_logic_vector(7 downto 0);  -- Read  Data

-- レジスタ
signal I_DR  : std_logic_vector(7 downto 0);  -- DR
signal I_OPR : std_logic_vector(7 downto 0);  -- OPR
signal I_AR  : std_logic_vector(7 downto 0);  -- AR
signal I_MPC : std_logic_vector(7 downto 0);  -- MPC
signal I_OP  : std_logic_vector(3 downto 0);  -- IR の OP
signal I_RD  : std_logic_vector(1 downto 0);  -- IR の Rd
signal I_RX  : std_logic_vector(1 downto 0);  -- IR の Rx

-- フラグ
signal I_C   : std_logic;                     -- Carry
signal I_S   : std_logic;                     -- Sign
signal I_Z   : std_logic;                     -- Zero
signal I_EI  : std_logic;                     -- Enable Interrupt

-- 内部配線
signal I_ALUT: std_logic_vector(8 downto 0);  -- ALU の内部で使用
signal I_ALU : std_logic_vector(7 downto 0);  -- ALU の出力

signal I_AC  : std_logic;                     -- ALU の Carry 出力
signal I_AZ  : std_logic;                     -- ALU の Zero  出力
signal I_AS  : std_logic;                     -- ALU の Sign  出力

signal I_JMP : std_logic;                     -- 機械語のジャンプ条件が成立
signal I_MPCLD : std_logic;                   -- MPC の値を変化
signal I_MPCAB : std_logic;                   -- MPC の入力選択

-- デコードROMの入出力
```

```
signal I_DADDR : std_logic_vector(7 downto 0);
signal I_DCODE : std_logic_vector(7 downto 0);

-- マイクロコードROMの出力
signal I_MCODE : std_logic_vector(31 downto 0);

-- マイクロコード
signal M_ALU : std_logic_vector(3 downto 0);
signal M_RR  : std_logic_vector(1 downto 0);
signal M_WR  : std_logic_vector(1 downto 0);
signal M_LR  : std_logic;
signal M_LF  : std_logic;
signal M_LA  : std_logic;
signal M_LO  : std_logic;
signal M_LD  : std_logic;
signal M_LI  : std_logic;
signal M_HL  : std_logic;
signal M_ER  : std_logic;
signal M_STI : std_logic;
signal M_CLI : std_logic;
signal M_RW  : std_logic;
signal M_MR  : std_logic;
signal M_IR  : std_logic;
signal M_JP  : std_logic_vector(2 downto 0);
signal M_JA  : std_logic_vector(7 downto 0);

component TEC_MROM
  port (
    P_CLK   : in  std_logic;
    P_RESET : in  std_logic;
    P_ADDR  : in  std_logic_vector(7 downto 0);
    P_DOUT  : out std_logic_vector(31 downto 0)
  );
end component;

component TEC_DROM
  port (
    P_CLK   : in  std_logic;
    P_RESET : in  std_logic;
    P_ADDR  : in  std_logic_vector(7 downto 0);
    P_DOUT  : out std_logic_vector(7 downto 0)
  );
end component;

begin
  -- マイクロコードＲＯＭ
  mrom0: TEC_MROM
     port map (P_CLK   => P_CLK,
               P_RESET => P_RESET,
                                    P_ADDR  => I_MPC,
               P_DOUT  => I_MCODE);

  -- デコードＲＯＭ
  drom0: TEC_DROM
     port map (P_CLK   => P_CLK,
               P_RESET => P_RESET,
               P_ADDR  => I_DADDR,
               P_DOUT  => I_DCODE);

  -- マイクロコード入力
  M_ALU <= I_MCODE(31 downto 28);
  M_RR  <= I_MCODE(27 downto 26);
  M_WR  <= I_MCODE(25 downto 24);
  M_LR  <= I_MCODE(23);
  M_LF  <= I_MCODE(22);
  M_LA  <= I_MCODE(21);
```



```
  M_LO  <= I_MCODE(20);
  M_LD  <= I_MCODE(19);
  M_LI  <= I_MCODE(18);
  M_HL  <= I_MCODE(17);
  M_ER  <= I_MCODE(16);
  M_STI <= I_MCODE(15);
  M_CLI <= I_MCODE(14);
  M_RW  <= I_MCODE(13);
  M_MR  <= I_MCODE(12);
  M_IR  <= I_MCODE(11);
  M_JP  <= I_MCODE(10 downto 8);
  M_JA  <= I_MCODE(7 downto 0);

  -- コントロールバス出力
  P_LI  <= M_LI;
  P_HL  <= M_HL;
  P_ER  <= M_ER;
  P_RW  <= M_RW;
  P_MR  <= M_MR;
  P_IR  <= M_IR;

  -- データバス出力
  P_DOUT <= I_OPR;

  -- パネルのLEDへの出力
  P_C   <= I_C;
  P_S   <= I_S;
  P_Z   <= I_Z;

  -- パネル用にレジスタを出力
  P_G0D <= I_G0;
  P_G1D <= I_G1;
  P_G2D <= I_G2;
  P_SPD <= I_SP;
  P_PCD <= I_PC;

  -- IR の制御
  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_OP <= "0000";
      I_RD <= "00";
      I_RX <= "00";
    elsif (P_CLK' event and P_CLK='1') then
      if (M_LI='1') then
        I_OP <= P_DIN(7 downto 4);
        I_RD <= P_DIN(3 downto 2);
        I_RX <= P_DIN(1 downto 0);
      end if;
    end if;
  end process;

  -- DR の制御
  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_DR <= "00000000";
    elsif (P_CLK' event and P_CLK='1') then
      if (M_LD='1') then
        I_DR <= P_DIN;
      end if;
    end if;
  end process;

  -- OPR の制御
  process(P_CLK, P_RESET)
```

```
  begin
    if (P_RESET='0') then
      I_OPR <= "00000000";
    elsif (P_CLK' event and P_CLK='1') then
      if (M_LO='1') then
        I_OPR <= I_ALU;
      end if;
    end if;
  end process;

  -- AR の制御
  P_ADDR <= I_AR;
  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_AR <= "00000000";
    elsif (P_CLK' event and P_CLK='1') then
      if (M_LA='1') then
        I_AR <= I_ALU;
      end if;
    end if;
  end process;

  -- FLAG の制御
  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_C <= '0';
      I_Z <= '0';
      I_S <= '0';
    elsif (P_CLK' event and P_CLK='1') then
      if (M_LF='1') then
        I_C <= I_AC;
        I_Z <= I_AZ;
        I_S <= I_AS;
      end if;
    end if;
  end process;

  -- 割り込みの制御
  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_EI <= '0';
    elsif (P_CLK' event and P_CLK='1') then
      if (M_STI='1') then
        I_EI <= '1';
      elsif (M_CLI='1') then
        I_EI <= '0';
      elsif (M_ALU = ALU_INF) then
        I_EI <= I_DR(7);
      end if;
    end if;
  end process;

  -- MPC の制御
  I_DADDR <= (I_OP & I_RD & I_RX);
  process(I_RD, I_C, I_Z, I_S)
  begin
    case I_RD is
      when "00"   => I_JMP <= '1';
      when "01"   => I_JMP <= I_Z;
      when "10"   => I_JMP <= I_C;
      when others => I_JMP <= I_S;
    end case;
  end process;
```



```
  process(M_JP, I_RX, I_JMP, P_INTR, P_STOP, I_EI)
  begin
    case M_JP is
      when JMP_NO  => I_MPCLD <= '0';           -- No
                      I_MPCAB <= 'X';
      when JMP_ALL => I_MPCLD <= '1';           -- JMP
                      I_MPCAB <= '0';
      when JMP_OP  => I_MPCLD <= '1';           -- JOP
                      I_MPCAB <= '1';
      when JMP_INT => I_MPCLD <= (P_STOP or (P_INTR and I_EI)); -- Jcc(INT)
                      I_MPCAB <= '0';
      when JMP_STP => I_MPCLD <= P_STOP;        -- Jcc(STP)
                      I_MPCAB <= '0';
      when JMP_JP  => I_MPCLD <= I_JMP;         -- Jcc(JP)  Jmp==1
                      I_MPCAB <= '0';
      when JMP_NJP => I_MPCLD <= not I_JMP;     -- Jcc(NJ)  Jmp==0
                      I_MPCAB <= '0';
      when others  => if (I_RX="00") then       -- Jcc(DI)  Rx==0
                        I_MPCLD <='1';
                        I_MPCAB <= '0';
                      else
                        I_MPCLD <='0';
                        I_MPCAB <= 'X';
                      end if;
    end case;
  end process;

  process(P_CLK, P_RESET)
  begin
    if (P_RESET='0') then
      I_MPC <= "00000000";
    elsif (P_CLK' event and P_CLK='1') then
      if (I_MPCLD='1') then
        if (I_MPCAB='0') then
          I_MPC <= M_JA;
        else
          I_MPC <= I_DCODE;
        end if;
      else
        I_MPC <= I_MPC + 1;
      end if;
    end if;
  end process;

  -- レジスタファイルの制御
  -- 書き込みレジスタの決定
  process(M_WR, M_LR, P_WRITE, I_RD, I_RX, P_SEL)
  begin
    if (M_LR='1') then
      case M_WR is
        when REG_SP => I_WRA <= "011";
        when REG_PC => I_WRA <= "100";
        when REG_RD => I_WRA <= '0' & I_RD;
        when others => I_WRA <= '0' & I_RX;
      end case;
    elsif (P_WRITE='1') then
      I_WRA <= P_SEL;
    else
      I_WRA <= "111";                           -- 書き込まない
    end if;
  end process;

  -- 書き込みデータ
  process(M_LR, I_ALU, P_PND)
  begin
```

```
    if (M_LR='1') then
      I_WRD <= I_ALU;
    else
      I_WRD <= P_PND;
    end if;
  end process;

  -- 書き込み制御
  process(P_CLK, P_RESET, P_MODE)
  begin
    if (P_CLK' event and P_CLK='1') then
      if (P_RESET='0') then              -- 非同期RESETではI_PC[7]がNG
        I_G0 <= "00000000";
        I_G1 <= "00000000";
        I_G2 <= "00000000";
        I_SP <= "00000000";
        I_PC <= P_MODE & "0000000";      -- DEMO MODE(2010.05.28)
      elsif (I_WRA="000") then I_G0 <= I_WRD;
      elsif (I_WRA="001") then I_G1 <= I_WRD;
      elsif (I_WRA="010") then I_G2 <= I_WRD;
      elsif (I_WRA="011") then I_SP <= I_WRD;
      elsif (I_WRA="100") then I_PC <= I_WRD;
      end if;
    end if;
  end process;

  -- 読みだしレジスタ番号
  process(M_RR, I_RD, I_RX)
  begin
    case M_RR is
      when REG_SP => I_RRA <= "011";
      when REG_PC => I_RRA <= "100";
      when REG_RD => I_RRA <= '0' & I_RD;
      when others => I_RRA <= '0' & I_RX;
    end case;
  end process;

  -- 読みだし制御
  process(I_G0, I_G1, I_G2, I_SP, I_PC, I_RRA)
  begin
    case I_RRA is
      when "000"  => I_RRD <= I_G0;
      when "001"  => I_RRD <= I_G1;
      when "010"  => I_RRD <= I_G2;
      when "011"  => I_RRD <= I_SP;
      when others => I_RRD <= I_PC;
    end case;
  end process;

  -- ALU の制御
  I_ALU <= I_ALUT(7 downto 0);

  process(I_RRD,I_DR,M_ALU,I_EI,I_C,I_S,I_Z,I_RX)
  begin
    case M_ALU is
      when ALU_ZERO => I_ALUT <= "000000000";                  -- Zero
      when ALU_A    => I_ALUT <= ('0' & I_RRD);                -- A
      when ALU_B    => I_ALUT <= ('0' & I_DR);                 -- B
      when ALU_ADD  => I_ALUT <= ('0' & I_RRD) + I_DR;         -- ADD
      when ALU_SUB  => I_ALUT <= ('0' & I_RRD) - I_DR;         -- SUB
      when ALU_AND  => I_ALUT <=  '0' & (I_RRD and I_DR);      -- AND
      when ALU_OR   => I_ALUT <=  '0' & (I_RRD or  I_DR);      -- OR
      when ALU_XOR  => I_ALUT <=  '0' & (I_RRD xor I_DR);      -- XOR
      when ALU_INC  => I_ALUT <= ('0' & I_RRD) + 1;            -- INC
      when ALU_DEC  => I_ALUT <= ('0' & I_RRD) - 1;            -- DEC
      --when ALU_ADC  => I_ALUT <= ('0' & I_RRD) + I_DR + I_C;    -- ADC
```



```
      when ALU_OUTF => I_ALUT <= ('0' & I_EI & "0000" & I_C & I_S & I_Z);
      when ALU_SFT  =>
        if (I_RX="10") then                        -- SRA
          I_ALUT(6 downto 0) <= I_RRD(7 downto 1);
          I_ALUT(7)          <= I_RRD(7);
          I_ALUT(8)          <= I_RRD(0);
        elsif (I_RX="11") then                     -- SRL
          I_ALUT(6 downto 0) <= I_RRD(7 downto 1);
          I_ALUT(7)          <= '0';
          I_ALUT(8)          <= I_RRD(0);
        else                                       -- SLA, SLL
          I_ALUT(8 downto 1) <= I_RRD(7 downto 0);
          I_ALUT(0)          <= '0';
        end if;
      when others   => I_ALUT <= "XXXXXXXXX";
    end case;
  end process;

  -- ALU のフラグ出力
  process (I_ALUT, M_ALU, I_DR)
  begin
    if (M_ALU = ALU_INF) then
      I_AZ <= I_DR(0);
      I_AS <= I_DR(1);
      I_AC <= I_DR(2);
    else
      if (I_ALUT(7 downto 0) = 0) then
        I_AZ <= '1';
      else
        I_AZ <= '0';
      end if;
      I_AS <= I_ALUT(7);
      I_AC <= I_ALUT(8);
    end if;
  end process;

end RTL;
```