```
----------------------------------------------------------------------------------
-- Company:         UTCN
-- Engineer:
--
-- Create Date:     15:57:46 04/11/2015
-- Design Name:     proc_RISC
-- Module Name:     proc_RISC - Behavioral
-- Project Name:    proc_RISC
-- Target Devices:
-- Tool versions:   Vivado 2016.4
-- Description:     Modul principal al procesorului RISC
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity proc_RISC is
   Generic (DIM_MI : INTEGER := 256;          -- dimensiunea memoriei de instructiuni (cuvinte)
            DIM_MD : INTEGER := 256);         -- dimensiunea memoriei de date (cuvinte)
   Port ( Clk      : in  STD_LOGIC;
          Rst      : in  STD_LOGIC;
          AdrInstr : out STD_LOGIC_VECTOR (31 downto 0);
          Instr    : out STD_LOGIC_VECTOR (31 downto 0);
          Data     : out STD_LOGIC_VECTOR (31 downto 0);
          RA       : out STD_LOGIC_VECTOR (31 downto 0);
          RB       : out STD_LOGIC_VECTOR (31 downto 0);
          F        : out STD_LOGIC_VECTOR (31 downto 0);
          ZF       : out STD_LOGIC;
          CF       : out STD_LOGIC);
end proc_RISC;

architecture Behavioral of proc_RISC is

   signal iCCEX       : STD_LOGIC_VECTOR (1 downto 0) := "00";
   signal sRID        : STD_LOGIC_VECTOR (21 downto 0) := (others => '0');
   signal iRID        : STD_LOGIC_VECTOR (21 downto 0) := (others => '0');
   signal sREX        : STD_LOGIC_VECTOR (6 downto 0) := (others => '0');
   signal sRI         : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
```

```
    signal sConst       : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');

    signal SSalt        : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal CSalt        : STD_LOGIC := '0';
    signal SelC         : STD_LOGIC := '0';

    signal DoutRegWr    : STD_LOGIC:= '0';
    signal DoutAdrD     : STD_LOGIC_VECTOR(3 downto 0):= (others => '0');
    signal DoutMxD      : STD_LOGIC_VECTOR(1 downto 0):= (others => '0');
    signal DoutSSalt    : STD_LOGIC_VECTOR(1 downto 0):= (others => '0');
    signal DoutCSalt    : STD_LOGIC:= '0';
    signal DoutMemWr    : STD_LOGIC:= '0';
    signal DoutOpUAL    : STD_LOGIC_VECTOR(3 downto 0):= (others => '0');
    signal DoutMxA      : STD_LOGIC:= '0';
    signal DoutMxB      : STD_LOGIC:= '0';
    signal EnInstr      : STD_LOGIC:= '0';

    signal x1, x2       : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal Q            : STD_LOGIC;



-- Semnale pentru interconectarea modulelor
    signal sMUXA        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMUXB        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMUXC        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMUXD        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sPC          : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal PCplus       : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMI          : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sPCIF        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sPCID        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sPCEX        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sRCONST      : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sRDoutA      : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sRDoutB      : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal AdrSalt      : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sUAL         : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMD          : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sFEX         : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sMDEX        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sCCEX        : STD_LOGIC_VECTOR (1  downto 0) := (others => '0');
    signal sSGTE        : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
    signal sSLT         : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');
```

```
   signal NxorV       : STD_LOGIC := '0';
   signal NxnorV      : STD_LOGIC := '0';
   signal V           : STD_LOGIC := '0';
   signal C           : STD_LOGIC := '0';
   signal N           : STD_LOGIC := '0';
   signal Z           : STD_LOGIC := '0';

-- Semnale de comanda
   signal RegWr       : STD_LOGIC := '0';
   signal MemWr       : STD_LOGIC := '0';
   signal OpUAL       : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
   signal Sh          : STD_LOGIC_VECTOR (4 downto 0) := (others => '0');
   signal MxA         : STD_LOGIC := '0';
   signal MxB         : STD_LOGIC := '0';
   signal MxC         : STD_LOGIC_VECTOR (1 downto 0) := (others => '0');
   signal MxD         : STD_LOGIC_VECTOR (1 downto 0) := (others => '0');
   signal AdrSA       : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
   signal AdrSB       : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
   signal AdrD        : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');

-- Semnale pentru eliminarea hazardului de control prin predictia salturilor
   signal sMI_hazard  : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');

begin
   MI:        entity WORK.mem_instr   port map (Clk => Clk, Rst => Rst, Adr => sPC (7 downto
0), Data => sMI);
   R:         entity WORK.set_reg     port map (Clk => Clk, Rst => Rst, WE => RegWr, AdrA =>
AdrSA, AdrB => AdrSB,
                                              AdrD => AdrD, Din => sMUXD, DoutA => sRDoutA, DoutB =>
sRDoutB);
   UAL:       entity WORK.unit_aritm  port map (X => sMUXA, Y => sMUXB, Sel => OpUAL, Sh
=> Sh, F => sUAL, V => V, C => C, N => N, Z => Z);
   MD:        entity WORK.mem_date    port map (Clk => Clk, Rst => Rst, WE => MemWr, Adr
=> sMUXA (7 downto 0), Din => sMUXB, Dout => sMD);



   INCPC:     PCplus <= sPC + 1;
   ADDPC:     AdrSalt <= sPCID + sMUXB;
   MUXA:      sMUXA <= sRDoutA when MxA = '0' else sPCID;
   MUXB:      sMUXB <= sRDoutB when MxB = '0' else sRCONST;
   MUXC:      with MxC select
              sMUXC <= PCplus when "00", AdrSalt when "01", sMUXA when "10", sPCEX when
others;
   MUXD:      with MxD select
```

```
                sMUXD <= sFEX when "00", sMDEX when "01", sSGTE when "10", sSLT when
others;

   bistabilD: process(Clk)
   begin
      if rising_edge(Clk) then
         if Rst = '1' then
            Q <= '1';
         else
            Q <= EnInstr;
         end if;
      end if;
   end process;


   PC: entity WORK.FDN port map (Clk, Rst, sMuxC, '1', sPC);
   PCIF: entity WORK.FDN port map (Clk, Rst, sPC, '1', sPCIF);
   PCID: entity WORK.FDN port map (Clk, Rst, sPCIF, '1', sPCID);
   PCEX: entity WORK.FDN port map (Clk, Rst, sPCID, '1', sPCEX);
   REX: entity WORK.FDN generic map (n => 7) port map (Clk, Rst, sRID(21 downto 15), '1',
sREX);
   CCEX: entity WORK.FDN generic map (n => 2) port map (Clk, Rst, iCCEX,'1', sCCEX);
   FEX: entity WORK.FDN port map (Clk, Rst, sUAL, '1', sFEX);
   MDEX: entity WORK.FDN port map (Clk, Rst, sMD, '1', sMDEX);
   RI: entity WORK.FDN port map (Clk, Rst, sMI_hazard, '1', sRI);
   RID: entity WORK.FDN generic map (n => 22) port map (Clk, Rst, iRID, '1', sRID);
   RCONST: entity WORK.FDN port map (Clk, Rst, sConst, '1', sRConst);
   CONST: entity WORK.CONST port map (sRI(15 downto 0), SelC, SConst);
   DCDI: entity WORK.DCDI port map (sRI, DoutRegWr, DoutAdrD, DoutMxD, DoutSSalt,
DoutCSalt, DoutMemWr, DoutOpUAL, DoutMxA, DoutMxB, AdrSA, AdrSB, SelC);
   SELMUXC: entity WORK.SELMUXC port map (sRID(14 downto 13), sRID(12), Z, MxC);


   iRID<=(DoutRegWr and EnInstr) & (DoutAdrD) & (DoutMxD) & ((DoutSSalt(1) and EnInstr) &
(DoutSSalt(0) and EnInstr)) & (DoutCSalt) & (DoutMemWr and EnInstr) & (DoutOpUAL) &
(DoutMxA) & (DoutMxB) & sRI(4 downto 0);

   EnInstr<= not (MxC(1) or MxC(0));
   x1 <= (others => EnInstr);
   x2 <= (others => Q);

   sMI_hazard <= x1 and sMI and x2;

   MemWr <= sRID(11);
```

```
    OpUAL <= sRID (10 downto 7);
    MxA <= sRID(6);
    MxB <= sRID(5);
    Sh <= sRID(4 downto 0);

    iCCEX <= NxnorV & NxorV;

    NxorV  <= N xor V;
    NxnorV <= not NxorV;
    sSLT   <= x"0000_000" & b"000" & sCCEX(0);
    sSGTE  <= x"0000_000" & b"000" & sCCEX(1);

    RegWr <= sREX(6);
    AdrD <= sREX(5 downto 2);
    MxD <= sREX(1 downto 0);


-- Conectarea semnalelor la porturile de iesire
    AdrInstr <= sPCIF;
    Instr    <= sMI_hazard;
    Data     <= sMD;
    RA       <= sMUXA;
    RB       <= sMUXB;
    F        <= sUAL;
    ZF       <= Z;
    CF       <= C;

end Behavioral;
```