```
------------------------------------------------------------------
----------------
-- 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

-- 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';

-----------------------------------------------------------
    signal REXin, REXout: STD_LOGIC_VECTOR(6 downto 0) :=
"0000000";
    signal CCEXin , ssalt      : STD_LOGIC_VECTOR(1 downto 0) :=
"00";
    signal CCEXout , ridout3     : STD_LOGIC_VECTOR(1 downto 0) :=
"00";
    signal ridIN, ridOUT: std_logic_vector(21 downto 0) := (others
=> '0');
    signal sri, im_ex : std_logic_vector(31 downto 0);
    signal EnInstr, csalt, ridout1, q: std_logic := '0';
    signal ridout2: std_logic_vector(3 downto 0) := "0000";
    signal im: std_logic_vector(15 downto 0):= (others => '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);

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

    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;

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



    CCEXin(1) <= not(N or V);
    CCEXin(0) <= N or V;
    EnInstr <= not(MxC(1) or MxC(0));
    RIDin <= (EnInstr and RegWr) & AdrD & MxD & (EnInstr and
SSalt(1)) & (EnInstr and SSalt(0)) & CSalt & (EnInstr and MemWr) &
OpUAL & MxA & MxB & Sh;
    REXin <= RIDout1 & RIDout2 & RIDout3;
    RIDout1 <= RIDout(21);
    RIDout2 <= RIDout(20 downto 17);
    RIDout3 <= RIDout(16 downto 15);
    SSalt <= RIDout(14 downto 13);
    CSalt <= RIDout(12);
    MemWr <= RIDout(11);
    OpUAL <= RIDout(10 downto 7);
    IM <= sRI(15 downto 0);
    MxA <= RIDout(6);
    MxB <= RIDout(5);
    Sh <= RIDout(4 downto 0);
    RegWr <= REXout(6);
    AdrD <= REXout(5 downto 2);
    MxD <= REXout (1 downto 0);



-- PORT MAP FDN
PCIF: entity WORK.FDN generic map (n => 32)
      port map (Clk => Clk, D => sPC, CE => '1', Rst => Rst, Q =>
sPCIF);
```

```
PCID: entity WORK.FDN generic map (n => 32)
      port map (Clk => Clk, D => sPCIF, CE => '1', Rst => Rst, Q
=> sPCID);
PCEX: entity WORK.FDN generic map (n => 32)
      port map (Clk => Clk, D => sPCID, CE => '1', Rst => Rst, Q
=> sPCEX);
REX: entity WORK.FDN generic map (n =>  7)
     port map (Clk => Clk, D => REXin, CE => '1', Rst => Rst, Q =>
REXout);
CCEX: entity WORK.FDN generic map (n => 2)
      port map (Clk => Clk, D => CCEXin, CE => '1', Rst => Rst, Q
=> sCCEX);
FEX:  entity WORK.FDN generic map (n => 32)
      port map(Clk => Clk, D => sUAL, CE => '1', Rst => Rst, Q =>
sFEX);
MDEX: entity WORK.FDN generic map (n => 32)
      port map(Clk => Clk, D => sMD, CE => '1', Rst => Rst, Q =>
sMDEX);
RCONST: entity WORK.FDN     generic map (n => 32)
        port map (Clk => Clk, D => IM_EX, CE => '1', Rst => '1', Q
=> sRCONST);
RI: entity WORK.FDN
    generic map (n => 32) port map(Clk => Clk, D => sMI_hazard, CE
=> '1', Rst => Rst, Q => sRI);
RID: entity WORK.FDN generic map (n => 22)
     port map(Clk => Clk, D => RIDin, CE => '1', Rst => Rst, Q =>
RIDout);

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

       gen: for i in 0 to 31 generate
```

```vhdl
            sMI_hazard(i) <= Q and EnInstr and sMI(i);
        end generate gen;

end Behavioral;
```