Laborator 8

Proiectarea în VHDL a unui microprocesor pe 8-biți

1. Setul de registre (R0-R7)

1.1 Specificații

Să se proiecteze un set de registre care să întrunească următoarele specificații:

- 16 registre organizate în două bancuri, unul este bancul normal de lucru, iar celălalt este accesat pe durata unei întreruperi;
- setul de registre va fi proiectat ca şi o memorie dual-port, cu două ieşiri şi o intrare, toate pe 8-biți;
- setul de registre trebuie să permită ca într-un singur ciclu de tact să fie citite două locații din memorie, iar una dintre ele să fie scrisă.
- Setul de registre dual port (notat ***uP8_REG*** (vezi Figura 1)) va avea nevoie de trei biți de adrese pentru a adresa cele două bancuri a câte opt registre (notați intrările A_ADDR şi B_ADDR). Acestea activează registrele a căror conținut va fi citit prin intermediul ieşirilor notate A_OUT şi B_OUT.

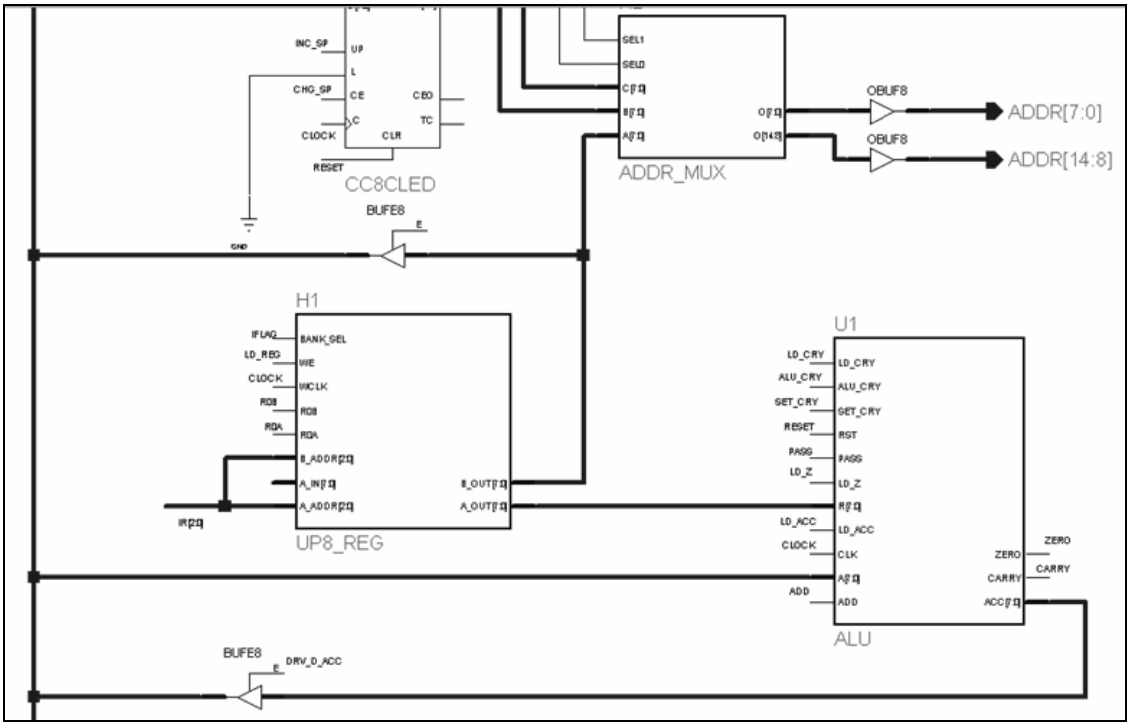

Figura 1. Setul de registre şi ALU

- De asemenea intrarea A_ADDR va selecta registrul care va fi activ pentru scrierea datelor prin intermediul intrării A_IN, date provenind de pe o magistrala internă D. Valoarea prezentă la intrarea A_IN va fi scrisă la următorul front crescător de tact (registrele sunt proiectate ca şi memorie sincronă), dacă la intrarea LD_REG semnalul are nivel logic „1”.
- Pe durata executării programelor microprocesorul are adesea nevoie de un registru în care să stocheze valori imediate şi adrese, pentru acest scop este selectat registrul R0. Modalitatea de accesare a acestui registru de către decodorul de instrucțiuni al microprocesorului este de a aplica un „1” logic la intrările R0A şi R0B, cea ce va duce la inhibarea intrării de adrese şi la forțarea adresei lui R0 la intrarea A_ADDR sau B_ADDR, vezi figura 2 ;
- O intrarea BANK_SEL trebuie să permită activarea la un moment dat doar a unuia din cele două bancuri de registre. Microprocesorul foloseşte un semnal IFLAG activ pe „1” logic pe durata executării unei subrutine de tratare a întreruperilor şi este folosit pentru a comuta între cele două bancuri de registre.

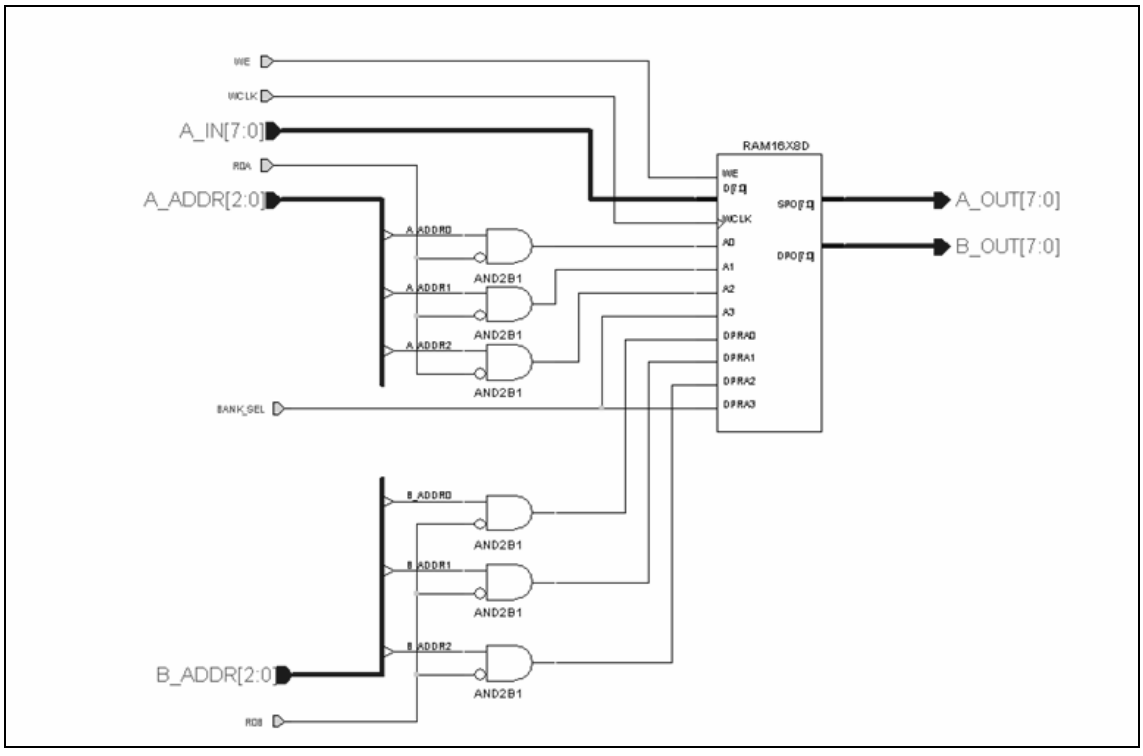

Figura 2. Schema detailată a setului de registre

Detaliile de proiectare cu blocuri în schematic, ale setului de registre sunt prezentate în Figura 2. Componenta principală a setului de registre este o memorie RAM sincronă dual port (RAM16X8D) selectată din lista de componente standard a mediului Xilinx. Bitul cel mai semnificativ al intrărilor de adrese pentru cele două bancuri de registre provine de la intrarea BANK_SEL. Pentru a forța adresele registrelor în zero se folosesc două seturi de porți logice ŞI controlate de semnalele R0A şi R0B.

1.2 Descrierea în VHDLa setului de registre

Descrierea în VHDL se va face plecând de la descrierea memorie RAM din laboratorul 6, figura 9.

```
entity reg16 is
  port (A_addr: in std_logic_vector (2 downto 0);
            B_addr:...;
            A_in: in std_logic_vector(7 downto 0);
        CLK, LD_reg, WE, BANK_sel, R0A, R0B: in std_logic;
            A_out: out std_logic_vector(7 downto 0);
            B_out: ...);
end entity reg16;
architecture behavioral of reg16 is

begin

Bank_A: process (CLk) is
      type ram_arrayA is array (0 to 7) of
            std_logic_vector(7 downto 0);
      variable memA: ram_arrayA;
          variable A_addr_var : integer;

    begin
     if clk'event and clk ='1' then
          if Bank_sel = '0' then
               if R0A ='1' and R0B='1' then
                    A_addr_var := 0;
               else
                    A_addr_var := Conv_integer (A_addr);
               end if;
               if WE = '0' then
                    A_out <= memA(A_Addr_var);
               elsif LD_reg = '1' then
          memA(A_Addr_var) := A_in;
                A_out <= memA(A_Addr_var);
               end if;
          else null;
          end if;
     end if;
```

```
   end process Bank_A;


Bank_B: process (CLk) ....—asemanator procesului Bank_A

   end process Bank_B;

     end architecture behavioral;
```

Figura 3. Cod VHDL pentru setul de registre

Codul VHDL din figura 3 cuprinde două procese, câte unul pentru fiecare banc de registre.

Completați secvențele de cod lipsă, verificați sintaxa şi simulați proiectul. Un exemplu de stimuli ce pot fi aplicați pentru testarea setului de registre este prezentat în figura 4.

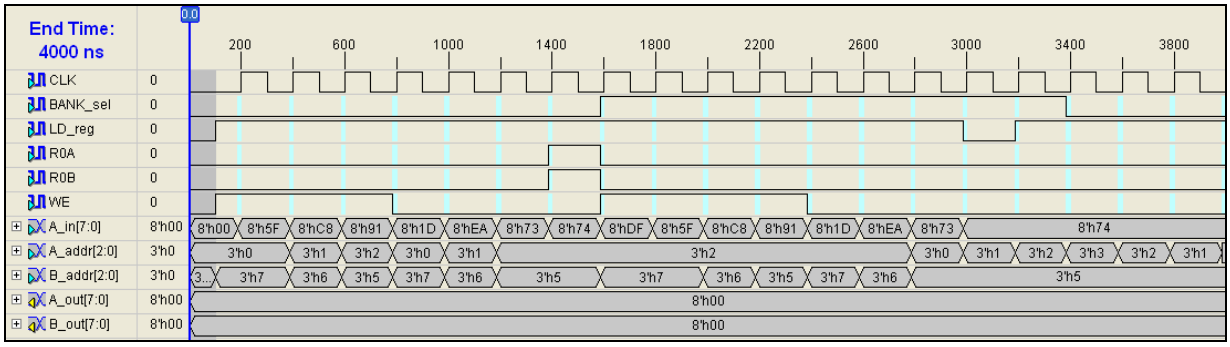

Figura 4. Stimuli pentru verificarea funcționării setului de registre

2. Unitatea aritmetică şi logică, registrul acumulator şi flagurile de stare

2.1 Specificații

Unitatea aritmetică şi logică (ALU) are rolul de a efectua toate operațiile logice (ŞI, SAU, SAU-Exclusiv etc.) şi aritmetice (adunare, scădere etc.) cu octeții de date. Rolul registrului acumulator (ACC) este acela de a stoca valorile intermediare rezultate în urma operațiilor făcute

de ALU. Registrul ACC este asemeni unei locații de memorie şi poate stoca un singur octet la un moment dat. Flagul sau fanionul de stare este reprezentat fizic printr-o celulă de memorie care are posibilitatea să stocheze la un moment dat un singur bit. În cazul microprocesorului descris la curs există două astfel de flaguri de stare şi anume flagul de transport sau de carry (C) şi flagul de zero (Z). Flagul de carry stochează valoarea de la ieşirea de transport (carry) a lui ALU, iar flagul Z este setat sau nu în funcție de rezultatul operațiilor efectuate de ALU.

Modulul ALU din Figura 1 trebuie să permită:

- Efectuarea operațiilor de ***adunare*** şi ***sau-exclusiv***;
- Modulul include şi acumulatorul şi flagurile de stare: carry (C), zero (Z);
- Operanzii (date sau adrese) pe 8-biți intră în ALU prin intermediul magistralelor R7...R0 şi A7...A0. Operanzii pot să provină de la setul de registre sau de pe magistrala internă D;
- Intrările de ADD şi PASS vor controla operațiile aritmetice care se efectuează cu operanzii mai sus amintiți conform tabelului T.1.

Tabelul T.1 Codificarea operațiilor aritmetice

PASS	**ADD**	**Operație ALU**
0	0	ACC<-ACC$REGA
0	1	ACC<-ACC + REGA + CARRY
1	X	ACC<- BUS-ul D

Detaliile în ceea ce priveşte structura ALU descrisă cu simboluri schematice sunt prezentate în Figura 5. Prin intermediul intrărilor ADD şi PASS sunt controlate o pereche de multiplexoare (H2 şi H5) fiecare cu intrări pe 8-biți. Aceste multiplexoare pot să aleagă între: octetul provenind de la intrarea A7...A0, ieşirea sumatorului ADD8 sau ieşirea din modulul XOR_8X8 care face operația SAU_EXCLUSIV bit-cu-bit. Când intrarea LD_ACC are valoarea „1” logic şi concomitent are loc o tranziție din „0” în „1” a semnalului de tact, modulul acumulator ACCUM0 va stoca valoarea de la ieşirea multiplexorului H2. Flagurile Z şi C îşi vor reîmprospăta de asemenea valoarea pe durata unei tranziții pozitive a semnalului de clock cu condiția ca semnalele LD_Z şi LD_CRY să aibă valoarea „1” logic. Flagul Z este întotdeauna încărcat cu valoarea de la ieşirea modulului ZEROTEST care face operația logică SAU-NU între biții rezultați în urma unui ŞI logic bit-cu-bit (vezi Figura 6). Octeții prinşi în operația ŞI bit-cu-bit provin de la setul de registre şi de pe magistrala D. Dacă intrarea ALU_CRY are valoarea „1” logic, flagul de transport C va stoca valoarea de la ieşirea de transport a modulului sumator (ieşirea CO a modulului ADD8), vezi Figura 7. Acesta permite propagarea transportului în operațiile aritmetice multi-octet.

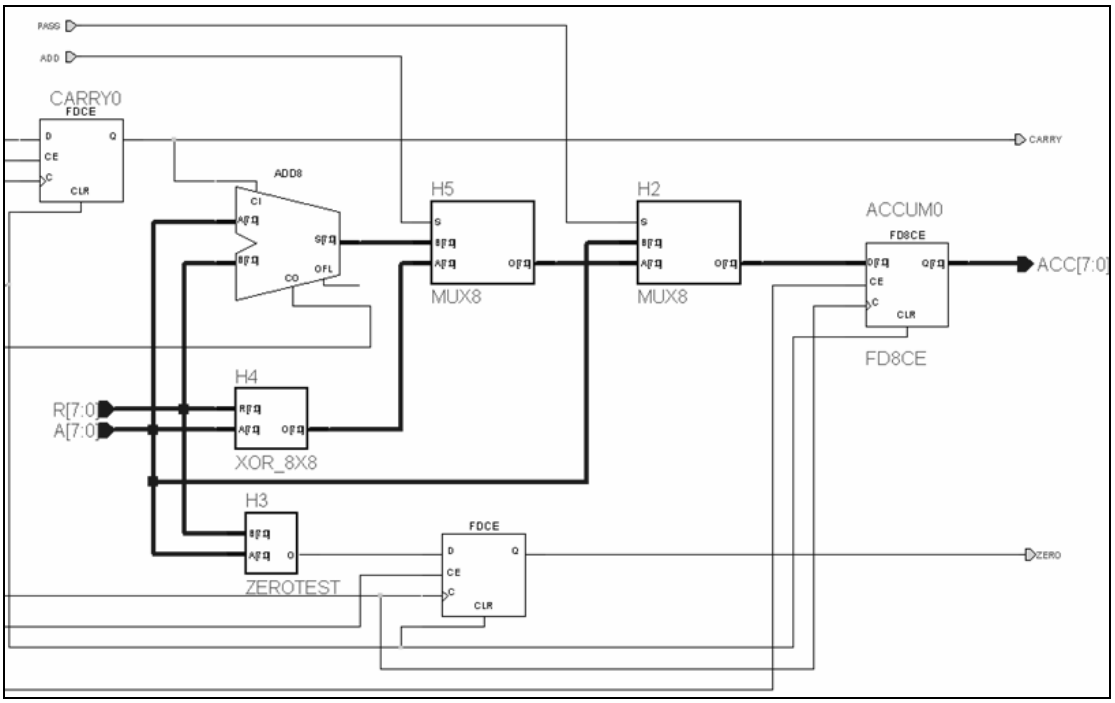

Figura 5.a Unitatea Logică şi Aritmetică

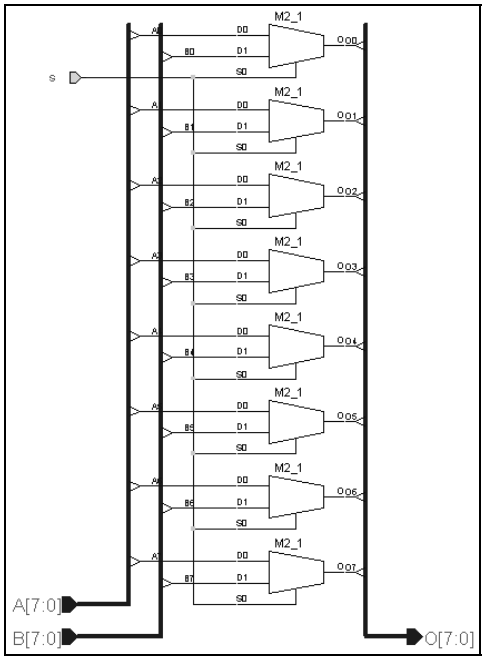

Figura 5.b Circuit detaliat MUX8

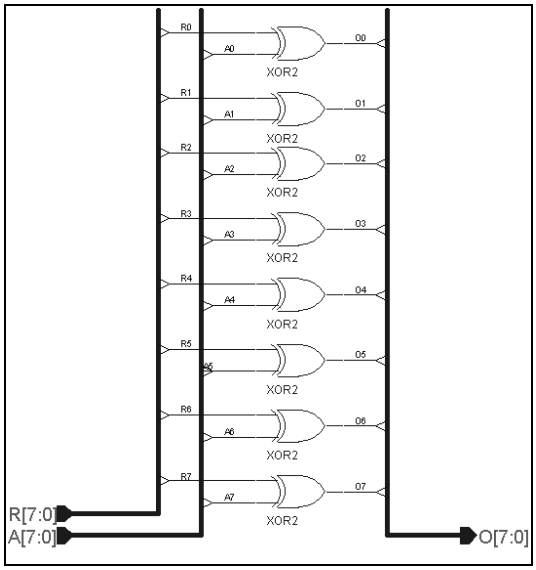

Figura 5.c Circuit detaliat XOR 8x8

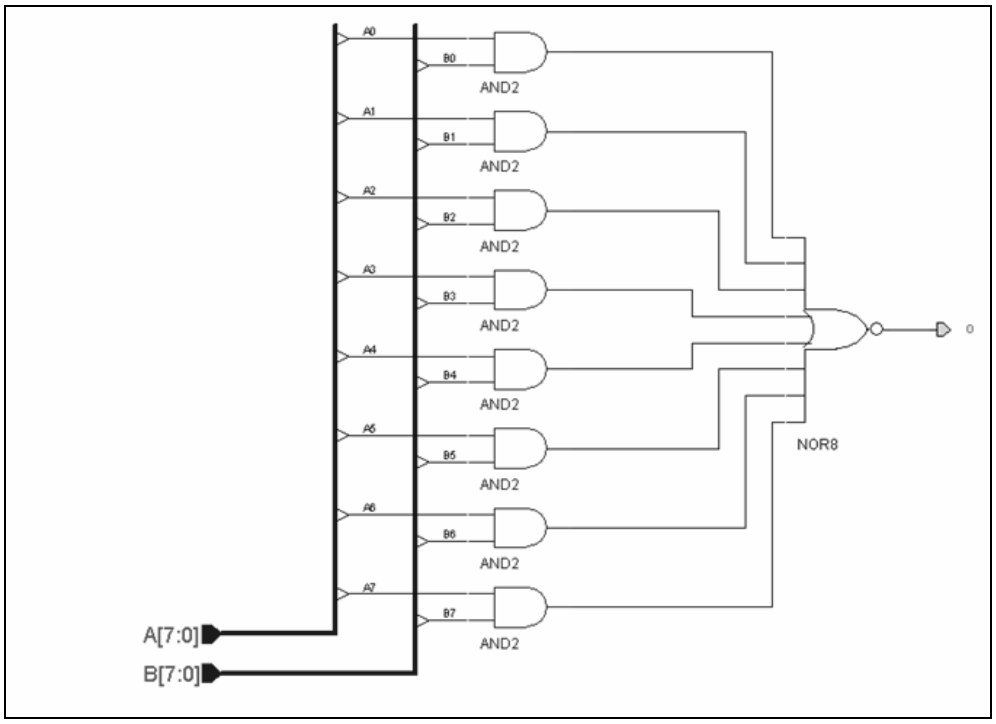

Figura 6 Modulul ZEROTEST

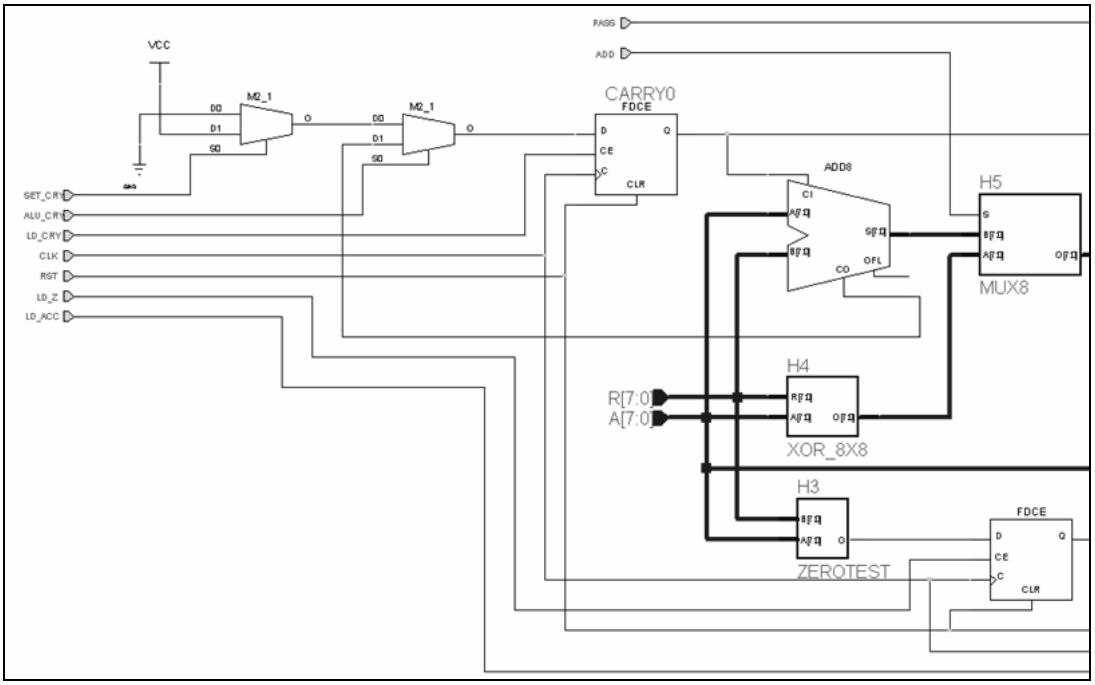

Figura 7 Circuit detailat corespunzător flag-ului de transport (Carry) C

Dacă intrarea ALU_CRY = „0” flagul C va fi încărcat cu valoarea de la intrarea SET_CRY care îl va seta în „0” sau în „1” logic.
O valoare de „1” logic la intrarea RESET va avea ca rezultat ştergerea conţinutului acumulatorului şi a celor două flaguri de stare. Ieşirile CARRY şi ZERO ale modulului ACC sunt asociate celor două flaguri. De asemenea ieşirea ACC7...ACC0 este asociată acumulatorului, setând semnalul DRV_D_ACC=”1” valoarea de la ieşirea acumulatorului va apărea pe magistrala internă D vezi Figura 2. Aceasta permite stocarea valorii provenind de la ACC în memoria externă, în setul de registre sau încărcarea ei din nou înapoi în ALU ca şi operand.

2.2 Descrierea în VHDL a unității logice şi aritmetice

Descrierea în VHDL (figura 8) se va face plecând de la descrierea memorie ALU din laboratorul 3, figura 12.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ALU is
port(A,R: in STD_LOGIC_VECTOR(7 downto 0);
LD_CRY, ALU_CRY, SET_CRY, RESET, PASS, LD_Z, LD_ACC, CLOCK, ADD:
in STD_LOGIC;
ZERO, CARRY: out STD_LOGIC;
ACC: out STD_LOGIC_VECTOR(7 downto 0));
end entity ALU;

architecture ALG of ALU is
signal A_sig, R_sig: UNSIGNED (8 downto 0):="000000000";
signal ACC_sig : UNSIGNED (8 downto 0):="000000000";
signal CARRY_sig: UNSIGNED (8 downto 0):="000000000";
signal CARRY_sig_out: std_logic;
signal Zero_sig: std_logic;
signal sel: std_logic_vector (1 downto 0);
begin
sel <= PASS&ADD;

process (RESET, CLOCK, PASS, ADD, R_sig, A_sig, CARRY_sig,
ACC_sig, zero_sig, sel, A, R )

begin

zero_sig <= (A(7)and R(7))or (A(6)and R(6))or (A(5)and R(5))or
(A(4)and R(4))or (A(3)and R(3))or
(A(2)and R(2))or (A(1)and R(1))or (A(0)and R(0));
```

```
A_sig  <= '0'& UNSIGNED (A);
R_sig <= '0'&UNSIGNED(R);


case  sel is
     when "00" => ACC_sig     <= '0' & UNSIGNED(A xor R);
     when "01" => ACC_sig     <= A_sig  + R_sig  + CARRY_sig;
     when others => ACC_sig   <= A_sig;
end case;

CARRY_sig_out<= ACC_sig (8);


if reset = '1' then
      CARRY_sig <= (others => '0');
      ZERO_sig <= '0';
      ACC_sig <= (others => '0');
      ACC <= (others => '0');
elsif clock'event and clock = '1' then
     if LD_Z = '1' then
     ZERO <= ZERO_sig;
     end if;
     if LD_CRY = '1' then
          if ALU_cry = '1' then
          carry_sig (0) <= Carry_sig_out;
               carry <= carry_sig_out;
          elsif ALU_cry = '0' then
               carry_sig (0) <= set_cry;
          end if;
     end if;
     if LD_ACC = '1' then
          ACC <= conv_std_logic_vector (ACC_sig(7 downto 0),8);
     end if;
end if;
end process;

end architecture ALG;
```

Figura 8. Cod VHDL pentru ALU

Verificați sintaxa, sintetizați şi simulați proiectul. Verificați dacă funcționarea corespunde specificațiilor enunțate în paragrafele anterioare.

Un exemplu de stimuli ce pot fi aplicați la intrarea unității logice şi aritmetice este prezentat în figura 9.

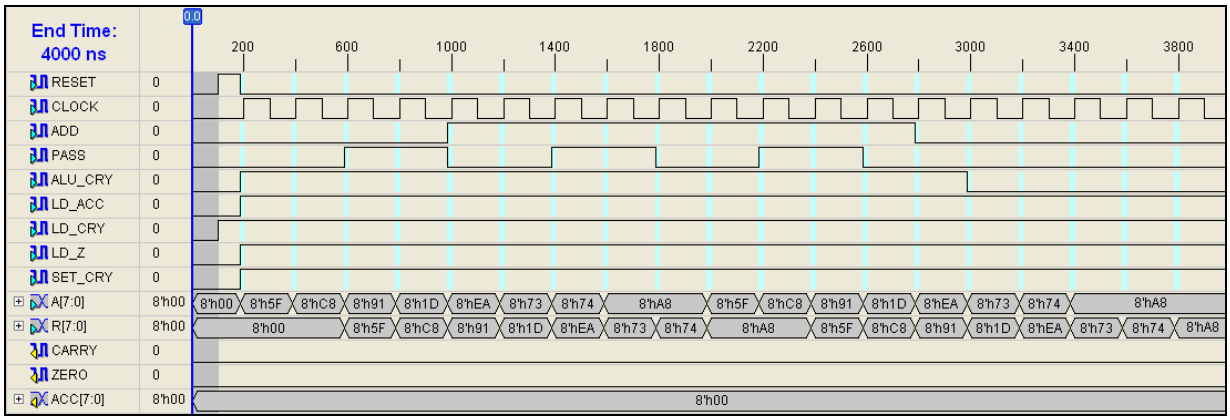

Figura 9. Stimuli pentru testarea ALU