Laborator 10

Proiectarea în VHDL a unui microprocesor pe 8-biți – partea a III-a

1. **Portul de intrare/ieşire**
2. **Registrul de instrucțiuni**
3. **Decodificatorul de instrucțiuni**
4. **Asamblarea proiectului**

1. Portul de intrare/ieşire

Există un singur port de intrare/ieşire (IOP), acesta are separate cele două magistrale, de intrare şi de ieşire, ambele fiind pe 8-biți. Modul de interconectare al componentele logice din care este alcătuit portul I/O este prezentat în figura 12.

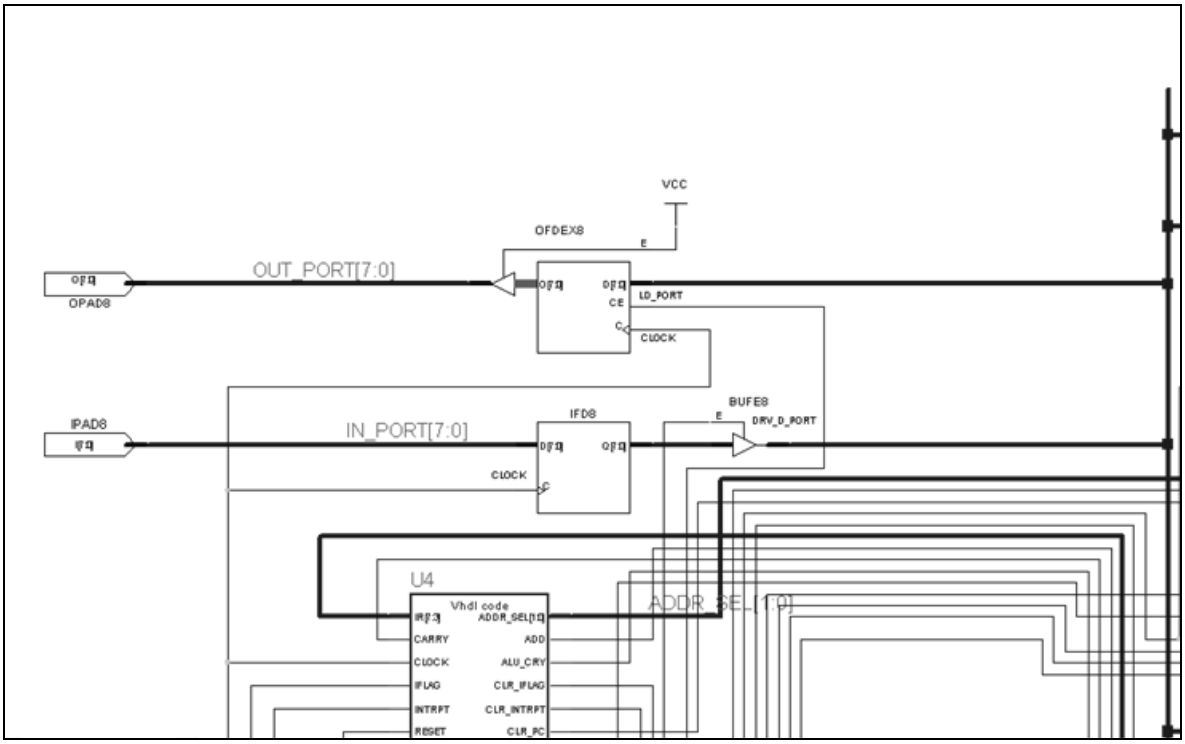

Figura 1 Portul de intrare/ieşire

1.1 Specificații

- Portul de ieşire trebuie descris ca şi registru, astfel încât furnizarea datelor de ieşire să se facă sincron cu semnalul de tact. Dacă o intrarea LD_PORT, a portului de ieşire este „1” logic, la un front crescător a semnalului de tact la de ieşire va apărea valoarea de pe magistrala internă de date D.
- Portul de intrare trebuie de asemenea descris ca şi registru. În acestea încarcându-se valoarea de la intrarea IN_PORT[7...0] la fiecare front crescător de tact. Valoarea de la ieşirea portului de intrare va apărea pe magistrala D numai când semnalul DRV_D_PORT este „1” logic şi numai sincron cu frontul pozitiv al semnalului de tact. Aceasta are ca efect sincronizarea portului de intrare cu celelalte blocuri ale microprocesorului *μP8*.

1.2 Descrierea în VHDL a portului de intrare/ieşire

Descrierea în VHDL se va face plecând de la descrierea registrelor din laboratorul 4.

```
entity port_out is
    Port ( clock : in  STD_LOGIC;
           load : in  STD_LOGIC;
           D : in  STD_LOGIC_VECTOR (7 downto 0);
           Q : out  STD_LOGIC_VECTOR (7 downto 0));
end port_out;

architecture Behavioral of port_out is
.....

end Behavioral;
```

Figura 2. Cod VHDL corespunzător portului de ieşire

Completați secvențele de cod lipsă ținând cont de specificații şi de blocul corespunzător port de intrare sau de ieşire, vezi diagrama din figura 1. Verificați sintaxa şi simulați proiectul.

În mod similar se va proceda şi pentru descrierea portului de intrare

2. Registrul de instrucțiuni

2.1 Specificații

Registrul de instrucțiuni (IR) este pe 8-biți şi are rolul de a stoca adresa instrucțiuni în curs de executare. În Figura 2 este prezentat registru de instrucțiuni modulul IR0 care este conectat direct la busul numit DATE.

- Registrul de instrucțiuni se încarcă cu toate valorile de pe magistrala DATE, pe care sunt prezente opcodurile instrucțiunilor de program stocate în RAM-ul extern.
- Încărcarea registrului de instrucțiuni are loc numai dacă semnalul LD_IR=”1” şi are loc o tranziție a semnalului de tact.
- Registrul de instrucțiuni este reinițializat când semnalul RESET este pe „1” logic.

Magistrala DATE menționată mai sus este bidirecțională şi asigură legătura cu memoria RAM externă. Valorile de pe magistrala internă D vor fi puse pe magistrala DATE când semnalul WRITE=”1”. Acest semnal comandă bufferul tristate OBUFE8. Datele din memoria RAM externă pot să ajungă pe magistrala D dacă semnalul de la decodificatorul de instrucțiuni DRV_DATA este „1” logic.

2.2 Descrirea în VHDL a registrului de instrucțiuni

Codul în VHDL al registrului de instrucțiuni se va descrie asemănător cu cel al porturilor de intrare/ieşire, conform exemplelor de registre in laboratorul 4. Verificați sintaxa codului VHDL şi simulați proiectul.

3. Decodificatorul de instrucțiuni

3.1 Specificații

Decodificatorul de instrucțiuni (ID) este componenta cea mai importantă şi cea mai complexă a unui microprocesor. ID are rolul de a interpreta instrucțiunea curentă care se află în IR (magistrală de intrare pe 8-biți). La interpretarea instrucțiunii curente va ține cont atât de starea flagurilor C şi Z cât şi de starea semnalului de la ieşirea circuitului de detectare a întreruperilor, astfel că în funcție de acestea decodificatorul de instrucțiuni va controla funcționarea celorlalte componente ale microprocesorului *μP8*.

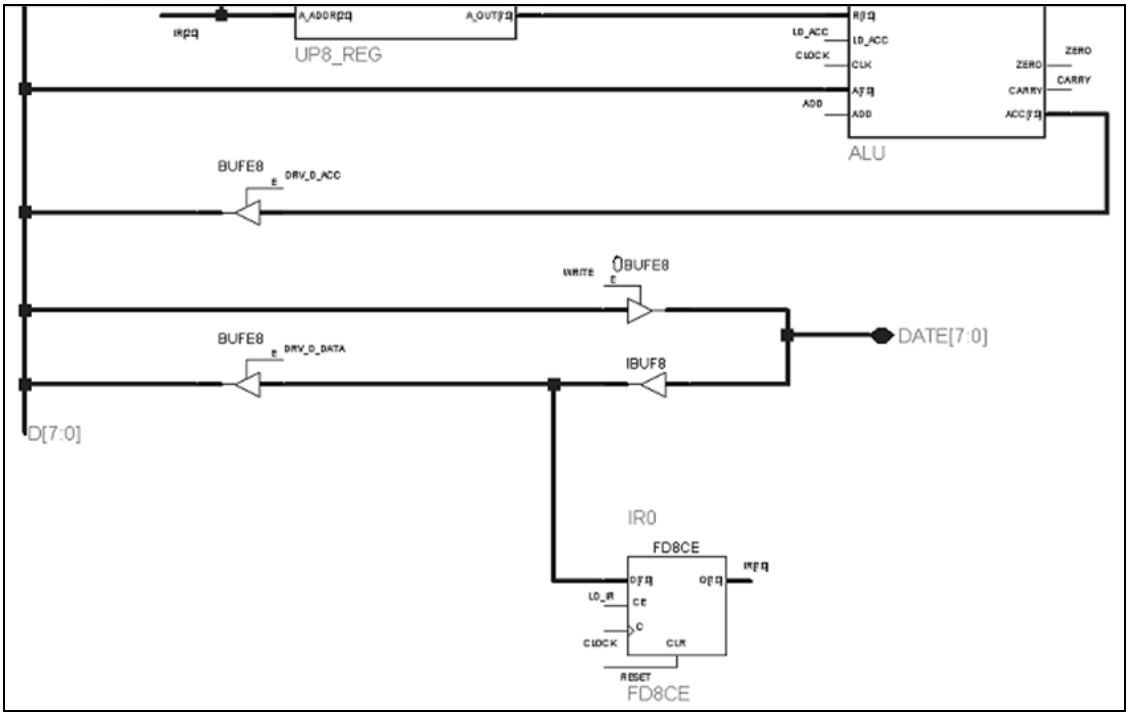

Figura 2 Registru de instrucțiuni şi interfața la magistrala externă de date

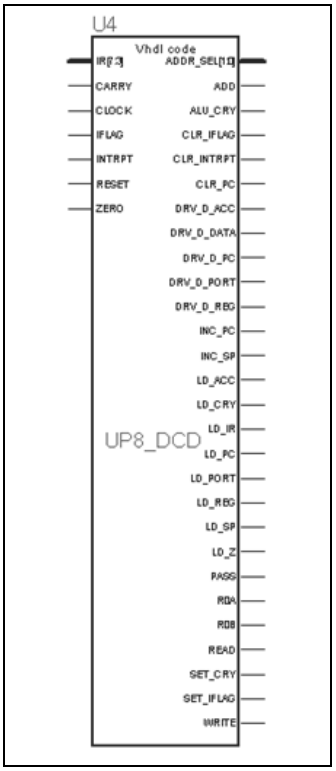

Figura 3 Decodificatorul de instrucțiuni

După cum se poate observa din figura 3 decodificatorul de instrucțiuni are 14 linii de intrare în funcție de care furnizează 29 de semnale de control pentru toate componentele microprocesorului tratate în acest paragraf. Modulul up8_IDCD din Figura 3 a fost creat prin importarea unui fişier VHDL sintetizat. Conținutul fişierului VHDL se poate vedea în tabelul T.1. În continuare vor fi tratate câteva instrucțiuni, restul codului pe urmă putându-se descifra uşor.
Pentru executarea unei instrucțiuni sunt necesare trei etape, aceste etape formează un **ciclu maşină**:

- Etapa FETCH (aducerea instrucțiuni) – microprocesorul citeşte instrucțiunea din zona de PROGRAM a memoriei RAM şi o încarcă în registrul de instrucțiuni IR;
- Etapa DECODE (decodificarea instrucțiuni) – microprocesorul (mai bine zis decodificatorul de adrese) trebuie să identifice tipul instrucțiuni care trebuie executată şi să aducă din memorie operanzii dacă este cazul;
- Etapa EXECUTE (executarea instrucțiuni) – microprocesorul trebuie să efectueze operațiile cu operanzii şi să stocheze rezultatul în memorie.

3.2 Codul VHDL corespunzător decodificatorului de instrucțiuni

Codul VHDL al decodificatorului de instrucțiuni începe cu declararea semnalelor de intrare (liniile 9-15) şi a semnalelor de ieşire (liniile 16-45). În linia 51 se declară ca şi semnale interne (acestea vor fi sintetizate ca şi bistabile) stările microcontrolerului, după care ca şi constante se vor declara valorile pe care le pot lua cele două stări: curr_st şi next_st (liniile 54-60). De asemenea tot ca şi valori constante vor fi declarate şi cele trei zone de memorie (liniile 62-65) şi setul de instrucțiuni (liniile 68-89).

Tabelul T.1 Codul VHDL al decodificatorului de instrucțiuni

```
1   -- UP8 controller -- V1.0
2   library IEEE;
3   use IEEE.std_logic_1164.all;
4   use IEEE.std_logic_unsigned.all;
5
6   entity uP8 is
7           PORT
8           (
9                   clock: in STD_LOGIC;
10                  reset: in STD_LOGIC;                    -- reset global
11                  ir: in STD_LOGIC_VECTOR (7 downto 3); -- registru de instrucțiuni (IR)
12                  carry: in STD_LOGIC;                    -- flag carry (C)
13                  zero: in STD_LOGIC;             -- flag zero (Z)
14                  intrpt: in STD_LOGIC;                   -- sesizare întreruperi
15                  iflag: in STD_LOGIC;                    -- înregistrare întreruperi
16                  read: out STD_LOGIC;                    -- 1 când se citeşte RAM
17                  write: out STD_LOGIC;                   -- 1 când se scrie RAM
18                  addr_sel: out STD_LOGIC_VECTOR (1 downto 0); -- 2=PROG;1=DATE;0=STIVA
19                  inc_pc: out STD_LOGIC;                  -- 1 pentru incrementare PC
20                  ld_pc: out STD_LOGIC;                   -- 1 pentru încărcare PC
21                  clr_pc: out STD_LOGIC;                  -- 1 pentru ştergere PC
22                  inc_sp: out STD_LOGIC;                  -- 0=--sp; 1=sp++
23                  ld_sp: out STD_LOGIC;                   -- 1 pentru schimbarea valorii din SP
24                  ld_ir: out STD_LOGIC;                   -- 1 pentru încărcare IR
25                  ld_port: out STD_LOGIC;                 -- 1 pentru încărcare port I/O
26                  drv_d_DATE: out STD_LOGIC; -- 1 când se pun valori pe D de pe busul DATE
27                  drv_d_acc: out STD_LOGIC;     -- 1 când se pun valori pe D din acumulator
```

```
28                              drv_d_pc: out STD_LOGIC;        -- 1 când se pun valori pe D din PC
29                              drv_d_port: out STD_LOGIC;      -- 1 când se pun valori pe D de la portul I/O
30                              drv_d_reg: out STD_LOGIC;       -- 1 când se pun valori pe D de la setul de registre
31                              ld_reg: out STD_LOGIC;                                  -- 1 când se încarcă setul de registre
32                              R0A: out STD_LOGIC;                                     -- 1 forțează adresa din reg. A la 0
33                              R0B: out STD_LOGIC;                                     -- 1 forțează adresa din reg. B la 0
34                              set_cry: out STD_LOGIC;         -- 0=resetează flagul de carry; 1=setează flagul de carry
35                              alu_cry: out STD_LOGIC;         -- 1 pentru ca în urma unei operații ALU carry să fie reîmrpospătat
36                              ld_cry: out STD_LOGIC;          -- 1 activare flag carry
37                              ld_z: out STD_LOGIC;            -- 1 activare flag zero
38                              add: out STD_LOGIC;             -- 0=XOR; 1=ADD
39                              pass: out STD_LOGIC;    -- 0=ieşire sumator trece în ACC; 1=operandul trece prin ALU la ACC
40                              ld_acc: out STD_LOGIC;          -- 1 activare acumulator
41                              set_iflag: out STD_LOGIC;       -- 1 setează pe 1 flag de întrerupere
42                              clr_iflag: out STD_LOGIC;       -- 1 ştergere flag întrerupere
43                              clr_intrpt: out STD_LOGIC       -- 1 resetare flip-flop întrerupere
44              );
45      end uP8;
46
47
48      ARCHITECTURE uP8_arch of uP8 is
49
50      SIGNAL curr_st, next_st: STD_LOGIC_VECTOR (2 downto 0);   -- uP8
51
52      -- stările uP8 -----
53      constant R0: STD_LOGIC_VECTOR (2 downto 0) := "000";        -- stare reset 0
54      constant R1: STD_LOGIC_VECTOR (2 downto 0) := "001";        -- stare reset 1
55      constant T0: STD_LOGIC_VECTOR (2 downto 0) := "011";        -- starea normală 0
56      constant T1: STD_LOGIC_VECTOR (2 downto 0) := "010";        -- starea normală 1
57      constant T2: STD_LOGIC_VECTOR (2 downto 0) := "110";        -- starea normală 2
58      constant T3: STD_LOGIC_VECTOR (2 downto 0) := "100";        -- starea normală 3
59      constant I1: STD_LOGIC_VECTOR (2 downto 0) := "111";        -- stare de tratare a întreruperilor 1
60
61      -- definirea zonelor de memorie -----
62      constant PROGRAM: STD_LOGIC_VECTOR (1 downto 0) := "11"; -- regiunea de PROGRAM
63      constant DATE:    STD_LOGIC_VECTOR (1 downto 0) := "00";   -- regiunea de DATE
64      constant STIVA:   STD_LOGIC_VECTOR (1 downto 0) := "01";   -- regiunea de STIVA
65
66      -- uP8 opcodurile instrucțiunilor -----
67      constant STORE_REG: STD_LOGIC_VECTOR (4 downto 0) := "00000";
68      constant STORE_INX: STD_LOGIC_VECTOR (4 downto 0) := "00001";
69      constant STORE_DIR: STD_LOGIC_VECTOR (4 downto 0) := "00010";
70      constant STORE_IND: STD_LOGIC_VECTOR (4 downto 0) := "00011";
71      constant LOAD_REG: STD_LOGIC_VECTOR (4 downto 0)  := "00100";
72      constant LOAD_INX: STD_LOGIC_VECTOR (4 downto 0)  := "00101";
73      constant LOAD_DIR: STD_LOGIC_VECTOR (4 downto 0)  := "00110";
74      constant LOAD_IND: STD_LOGIC_VECTOR (4 downto 0)  := "00111";
75      constant LOAD_IMM: STD_LOGIC_VECTOR (4 downto 0)  := "01000";
76      constant IN_REG: STD_LOGIC_VECTOR (4 downto 0)    := "01100";
77      constant OUT_OP: STD_LOGIC_VECTOR (4 downto 0)    := "01101";
78      constant XOR_OP: STD_LOGIC_VECTOR (4 downto 0)    := "10000";
79      constant ADD_OP: STD_LOGIC_VECTOR (4 downto 0)    := "10001";
80      constant TEST: STD_LOGIC_VECTOR (4 downto 0)      := "10010";
81      constant CLEAR_C: STD_LOGIC_VECTOR (4 downto 0)   := "10100";
82      constant SET_C: STD_LOGIC_VECTOR (4 downto 0)     := "10101";
83      constant JC: STD_LOGIC_VECTOR (4 downto 0)        := "11000";
84      constant JZ: STD_LOGIC_VECTOR (4 downto 0)        := "11001";
85      constant JUMP: STD_LOGIC_VECTOR (4 downto 0)      := "11010";
86      constant JSR: STD_LOGIC_VECTOR (4 downto 0)       := "11011";
87      constant RET: STD_LOGIC_VECTOR (4 downto 0)       := "11100";
88      constant RETI: STD_LOGIC_VECTOR (4 downto 0)      := "11101";
89
90      BEGIN
91
92      process(clock,next_st,reset)
93      begin
94              if reset='1' then           -- resetare asincronă
95                      curr_st <= R0;
96              elsif (clock'event and clock='1') then
97                      curr_st <= next_st;-- la următorul front crescător de clock treci la starea următoare (next_state)
98
```

```
99                  end if;
100     end process;
101
102     process(curr_st,ir,carry,zero,iflag,intrpt)
103     begin
104       -- inițializare ieşiri pentru a prevenii sintetizarea bistabilelor
105       read<='0';
106       write <= '0';
107       addr_sel <= PROGRAM;
108       inc_pc <= '0';
109       ld_pc <= '0';
110       clr_pc <= '0';
111       inc_sp <= '0';
112       ld_sp <= '0';
113       ld_ir <= '0';
114       ld_port <= '0';
115       drv_d_DATE <= '0';
116       drv_d_acc <= '0';
117       drv_d_pc <= '0';
118       drv_d_port <= '0';
119       drv_d_reg <= '0';
120       ld_reg <= '0';
121       R0A <= '0';
122       R0B <= '0';
123       set_cry <= '0';
124       alu_cry <= '0';
125       ld_cry <= '0';
126       ld_z <= '0';
127       add <= '0';
128       pass <= '0';
129       ld_acc <= '0';
130       set_iflag <= '0';
131       clr_iflag <= '0';
132       clr_intrpt <= '0';
133       next_st <= R0;
134       case curr_st is
135               when R0 =>                      -- tratare stare reset prin incrementare PC la adresa 0x02
136                       inc_pc <= '1';          -- incrementare PC la 0x01
137                       next_st <= R1;
138               when R1 =>                      -- terminare incrementare PC to 0x02
139                       inc_pc <= '1';          -- incrementare PC la 0x02
140                       next_st <= T0;          -- începere procese de aducere şi executare instrucțiuni
141               when T0 =>                      -- tratarea instrucțiunilor în desfăşurarea normală a programelor
142                       if (intrpt='1' and iflag='0') then    -- depistare întrerupere: stocare PC în STIVA
143                               set_iflag <= '1';         -- setare flag întrerupere pt. a semnaliza ca o întrerupere este în
144                                                                       --curs de executare
145                               drv_d_pc<='1'; addr_sel<=STIVA; write<='1';             -- salvează PC în STIVA
146                               ld_sp<='1'; inc_sp<='1';                                -- incrementează STIVA pointer
147                               next_st <= I1;          -- continuă cu starea care tratează întreruperile
148                       else – desfăşurarea obişnuită a programului: aducerea unui opcod (instrucțiune)
149                               addr_sel<=PROGRAM; read<='1';             -- citeşte din RAM, regiunea de PROGRAM
150                               drv_d_DATE<='1'; ld_ir<='1';    -- încarcă opcodul în registrul de instrucțiuni
151                               inc_pc<='1';                    -- incrementează program counterul
152                               next_st <= T1;                  -- mergi la starea următoare :executare instrucțiune
153                       end if;
154               when I1 =>                                                      -- tratare întreruperi
155                        clr_intrpt <= '1';                     -- ştergere întreruperea curentă
156                       drv_d_acc<='1'; addr_sel<=STIVA; write<='1';  -- salvează ACC în STIVA
157                         ld_sp<='1'; inc_sp<='1';                      -- incrementează indicatorul de STIVA
158                         clr_pc<='1';                                  -- resetare PC
159                         next_st <= T0;                                -- se începe citirea codului de tratare a întreruperii
160               when T1 =>              -- procesarea instrucțiunilor
161                       case ir(7 downto 3) is
162                               when IN_REG =>
163                                       drv_d_port<='1'; ld_reg<='1';-- registrele se încarcă cu valorile de la portul de
164                                                                               --intrare
165                                       next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
166
167                               when OUT_OP =>
168                                       drv_d_acc<='1'; ld_port<='1';     -- portul de ieşire se încarcă cu valoarea din ACC
169
```

```
170                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
171
172             when JSR =>             -- obține adresa subrutinei şi incrementează PC la următoarea instr.
173                     addr_sel<=PROGRAM; read<='1';                   -- citeşte adrs. subr. din  RAM
174                     drv_d_DATE<='1'; ld_reg<='1'; R0A<='1';   -- încarcă adrs. în reg. R0
175                     inc_pc<='1';
176                     next_st <= T2;          -- executarea instrucțiunii continuă şi în starea următoare
177
178             when RET =>             -- decrementează indicatorul stivă pentru a indica adrs. de reîntoarcere
179                     ld_sp<='1'; inc_sp<='0';            -- decrementează indicatorul de STIVA
180                     next_st <= T2;          -- executarea instrucțiunii continuă şi în starea următoare
181
182             when RETI =>-- dec. indic. de stivă pt. a indica locația unde a fost salvat ACC
183                     ld_sp<='1'; inc_sp<='0';            -- decrementează indicatorul de STIVA
184                     next_st <= T2;          -- executarea instrucțiunii continuă şi în starea următoare
185
186             when JUMP =>
187                     addr_sel<=PROGRAM; read<='1';               -- citeşte adresa de salt din RAM
188                     drv_d_DATE<='1'; ld_pc<='1';                -- încarcă-o în PC
189                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
190
191             when JC =>
192                     if carry='1' THEN               -- daca carry=1 încarcă PC cu adresa de salt
193                             addr_sel<=PROGRAM; read<='1';-- citeşte adresa de salt din RAM
194                             drv_d_DATE<='1'; ld_pc<='1';    -- încarcă-o în PC
195                     ELSE                                    -- altfel treci la următoarea instr.
196                             inc_pc<='1';                    -- incrementează PC
197                     end if;
198                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
199
200             when JZ =>
201                     if zero='1' THEN        -- dacă flagul zero este setat, încarcă PC cu adresa de salt
202                     addr_sel<=PROGRAM; read<='1';               -- citeşte adresa de salt din RAM
203                             drv_d_DATE<='1'; ld_pc<='1';    -- încarcă-o în PC
204                     ELSE                                    -- altfel treci la următoarea instrucțiune.
205                             inc_pc<='1';            -- incrementează  PC
206                     end if;
207                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
208
209             when LOAD_REG =>                -- încarcă ACC cu date din registre
210                     pass<='1'; ld_acc<='1'; drv_d_reg<='1';
211                     next_st <= T0;          -- terminat instrucțiune: treci la următoarea
212
213             when LOAD_INX => -- încarcă ACC cu date ale căror adresa este în registre
214                     addr_sel<=DATE; read<='1';      -- adresează memoria cu adresă din registre
215                     drv_d_DATE<='1'; pass<='1'; ld_acc<='1';-- stochează  RAM DATE în ACC
216                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
217             when LOAD_IMM =>                -- încarcă ACC cu valori din zona de PROGRAM
218                     addr_sel<=PROGRAM; read<='1'; drv_d_DATE<='1';-- citeşte datele din RAM
219                     pass<='1'; ld_acc<='1';             -- treci datele prin ALU în ACC
220                     inc_pc<='1';                    -- inc. PC la instr. următoare.
221                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
222             when LOAD_DIR =>                -- încarcă R0 cu o adresă aflată în RAM
223                     addr_sel<=PROGRAM; read<='1'; drv_d_DATE<='1'; -- adu adresa din RAM
224                     R0A<='1'; ld_reg<='1';              -- salvează-o in R0
225                     inc_pc<='1';                    -- inc. PC la instr. următoare
226                     next_st <= T2;          -- executarea instrucțiunii continuă şi în starea următoare
227
228             when LOAD_IND =>                -- încarcă R0 cu adresa unei adrese din RAM
229                     addr_sel<=PROGRAM; read<='1'; drv_d_DATE<='1';   -- adu adresa din RAM
230                     R0A<='1'; ld_reg<='1';-- salveaz-o in R0
231                     inc_pc<='1';                -- inc. PC la instr. următoare
232                     next_st <= T2;          -- executarea instrucțiunii continuă şi în starea următoare
233
234             when STORE_REG =>               -- salvează ACC într-un registru
235                     drv_d_acc<='1'; ld_reg<='1';
236                     next_st <= T0;          -- terminat instrucțiune: treci la următoarea
237             when STORE_INX =>               -- salvează ACC în memorie la adresa aflată în registru
238                     addr_sel<=DATE;     -- adresează RAM cu adresă din registru
239                     drv_d_acc<='1'; write<='1';         -- pune valoarea din ACC pe busul DATE
```

```
240                                     next_st <= T0;          -- terminat instrucțiune: treci la următoarea
241                             when STORE_DIR =>               -- adu adresa la care va fi stocat ACC
242                                     addr_sel<=PROGRAM; read<='1'; drv_d_DATE<='1';-- adu adresa din  RAM
243                                     R0A<='1'; ld_reg<='1';          -- încarcă adresa în R0
244                                     inc_pc<='1';            -- inc. PC la instr. următoare
245                                     next_st <= T2;          -- executarea instrucțiunii continuă și în starea următoare
246
247                             when STORE_IND =>-- adu adrs. din RAM care conține adrs. la care va fi stocat ACC
248                                     addr_sel<=PROGRAM; read<='1'; drv_d_DATE<='1';-- obține adresa din RAM
249                                     R0A<='1'; ld_reg<='1';                  -- stochează adrs. în R0
250                                     inc_pc<='1';                            --inc. PC la instr. următoare
251                                     next_st <= T2;          -- executarea instrucțiunii continuă și în starea următoare
252
253                             when ADD_OP =>                  -- ACC <- ACC + Registru
254                                     pass<='0'; add<='1'; ld_acc<='1'; drv_d_acc <= '1';
255                                     ld_cry<='1'; alu_cry<='1';      -- încarcă în flag  carry  ALU carry out
256                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
257                             when XOR_OP =>                          -- ACC <- ACC $ Registru
258                                     pass<='0'; add<='0'; ld_acc<='1'; drv_d_acc <= '1';
259                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
260                             when TEST =>   -- AND bit-cu-bit, testează conținutul ACC în funcție de un registru
261                                     ld_z <= '1'; drv_d_acc<='1';    -- dacă rezultatul testări este zero flagul zero va fi 1
262                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
263                             when SET_C =>                           -- set flag  carry pe 1
264                                     ld_cry<='1'; set_cry<='1';
265                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
266                             when CLEAR_C =>                         -- șterge flag  carry , 0
267                                     ld_cry<='1'; set_cry<='0';
268                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
269                             when others =>
270                     end case;
271             when T2 =>                                              -- procesarea instrucțiunilor (cont.)
272                     case ir(7 downto 3) is
273                             when JSR =>             -- salvează PC în STIVA și inc. SP
274                                     drv_d_pc<='1'; addr_sel<=STIVA; write<='1';             --salvează PC în STIVA
275                                     ld_sp<='1'; inc_sp<='1';                                --inc. indicatorul de  STIVA
276                                     next_st <= T3;          -- executarea instrucțiunii continuă și în starea următoare
277
278                             when RET =>                     -- încarcă adresa de reîntoarcere din subrutină în PC
279                                     addr_sel<=STIVA; read<='1';     --citește adresa de reîntoarcere din STIVA
280                                     drv_d_DATE<='1'; ld_pc<='1';    --încarcă adresa in PC
281                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
282                             when RETI =>                            -- reîncarcă ACC din stivă și decrementează STIVA
283                             addr_sel<=STIVA; read<='1';     --citește val. ACC din STIVA
284                                     drv_d_DATE<='1'; pass<='1'; ld_acc<='1';   --încarc-o în ACC
285                                     ld_sp<='1'; inc_sp<='0';                --decrementează indicatorul de STIVA
286                                     next_st <= T3; -- executarea instrucțiunii continuă și în starea următoare
287
288                             when LOAD_DIR =>        -- încarcă ACC cu date din RAM ale căror adresă se află în R0
289                                     R0B<='1'; addr_sel<=DATE; read<='1';--citește datele din RAM cu adrs. din R0
290                                     drv_d_DATE<='1'; pass<='1'; ld_acc<='1';   --încarcă valorile în ACC
291                                     next_st <= T0;          -- terminat instrucțiune: treci la următoarea
292                             when LOAD_IND => -- încarcă date din RAM în R0 de la adresa aflată în R0
293                     R0B<='1'; addr_sel<=DATE; read<='1';        --obține adresa din RAM
294                                     drv_d_DATE<='1'; R0A<='1'; ld_reg<='1';--stochează adresa în R0
295                                     next_st <= T3;          -- executarea instrucțiunii continuă și în starea următoare
296
297                             when STORE_DIR =>                       -- stochează ACC în RAM la adresa din R0
298                             R0B<='1'; addr_sel<=DATE; write<='1';       --adresa de RAM est adusă în R0
299                                     drv_d_acc<='1'; write<='1';     --valoarea din ACC este  trimisă în  RAM
300                                     next_st <= T0;                  -- terminat instrucțiune: treci la următoarea
301                             when STORE_IND =>               -- obține adresa la care se stochează ACC
302             R0B<='1'; addr_sel<=DATE; read<='1';       --adresa RAM este stocată în R0
303                                     drv_d_DATE<='1'; R0A<='1'; ld_reg<='1';   --stochează noua adresă în R0
304                     next_st <= T3;          -- executarea instrucțiunii continuă și în starea următoare
305                             when others =>
306                     end case;
307             when T3 =>              -- procesarea instrucțiunilor (cont.)
308                     case ir(7 downto 3) is
309                             when JSR =>                     -- încarcă adresa subrutinei în PC
```

```
310                 drv_d_reg<='1'; R0B<='1';                                   --citeşte adresa subrutinei din R0
311                                                 ld_pc<='1';--şi încarc-o în PC
312                                                 next_st <= T0;          -- terminat instrucţiune: treci la următoarea
313                                         when RETI =>            -- PC încărcat cu adrs. de reîntoarcere din intrpt., flag intrpt. şters
314                                           addr_sel<=STIVA; read<='1';--citeşte adresa de reîntoarcere din STIVA
315                                                   drv_d_DATE<='1'; ld_pc<='1';          --încarcă adresa în PC
316                                           clr_iflag<='1';--indică că s-a încheiat procesarea rutinei de tratare a întreruperi
317                                     next_st <= T0;                  -- terminat instrucţiune: treci la următoarea
318                                         when LOAD_IND =>  -- încarcă ACC cu date ale căror adresă este în R0
319                                           R0B<='1'; addr_sel<=DATE; read<='1';   --citeşte adresa de RAM în R0
320                                                   drv_d_DATE<='1'; pass<='1'; ld_acc<='1';          --încarcă valoarea în ACC
321                                                   next_st <= T0;                -- terminat instrucţiune: treci la următoarea
322                                         when STORE_IND =>               -- stochează ACC în RAM la adresa stocată în R0
323                                                 R0B<='1'; addr_sel<=DATE; write<='1';--adresa de RAM este încărcată în R0
324                                                   drv_d_acc<='1'; write<='1';  --trimite valoarea din ACC în RAM
325                             next_st <= T0;          -- terminat instrucţiune: treci la următoarea
326                                         when others =>
327                             end case;
328                 when others =>
329       end case;
330     end process;
331
332     end uP8_arch;
```

Arhitectura decodificatorului cuprinde descrierea a două procese. În primul proces se face „reinițializarea" stării curente cu starea următoare şi resetarea asincronă a decodificatorului (liniile 93-100). Cel de al doilea proces inițializează toate semnalele de ieşire ale decodificatorului (liniile 105-133), după care va fi descrisă funcționarea decodificatorului în fiecare din cele şapte stări. Stările R0 şi R1 vor fi activate numai în faza de reinițializare după ce a fost setat semnalul de reset. Stările T0, T1, T2, T3 vor fi active pe măsură ce sunt executate instrucțiunile. Pe durata stării T0 se verifică starea logică a pinului prin intermediul căruia se detectează întreruperile, dacă se detectează o întrerupere starea care va iniția tratarea acesteia va fi I1.

În liniile de cod 134-333 este descris modul în care se face tranziția între stări şi semnalele de ieşire care sunt afectate la fiecare tranziție. Când are loc un semnal de reset microprocesorul *μP8* va trece în starea R0 (linia 135, bistabilii care stochează stările vor fi setați cu valoarea 000 care este codul stării R0). După resetare PC este inițializat cu valoarea „0". În starea R0 PC se incrementează o dată după care se va trece în starea următoare (liniile 135-137). În starea R1, PC se va mai incrementa o dată (va conține adresa 02), iar automatul de stări (decodificatorul de instrucțiuni) va trece în starea T0 (138-141). În starea T0 sunt aduse instrucțiunile şi începe executarea programului. În următorul paragraf se va explica de ce este necesar ca programele să înceapă la adresa 02 şi se va prezenta modul de preluare a unei întreruperi.

Faptul că programele trebuie să înceapă de la adresa 02 are legătură tocmai cu nevoia de a procesa întreruperi. Astfel în starea T0 se verifică dacă a avut loc o întrerupere (linia 142, intrpt=1) şi dacă nu cumva o subrutină de tratare a întreruperilor este în curs de executare (iflag=0). Dacă condițiile prezentate mai sus sunt adevărate, semnalul *iflag* va fi setat pentru a indica că are loc executarea unei subrutine de tratare a întreruperilor (linia 143), adresa instrucțiunii curente va fi salvată în memoria RAM în zona de stivă (linia 146), după care stiva este incrementată (linia 147). În continuare decodificatorul de instrucțiuni va trece în starea I1. În această stare bistabili din circuitul

de sesizare a întreruperilor vor fi inițializați cu zero (linia 156, prin aceasta se dă posibilitatea microprocesorului dea sesiza o nouă întrerupere), apoi se salvează conținutul ACC în stivă (linia 157), se incrementează indicatorul de stivă (linia 158), se şterge conținutul PC (linia 159), după care la următoarea perioadă de clock se trece în starea T0 şi se va începe procesarea programelor începând cu adresa 0 din PC. Se poate observa că toate programele (serviciile) de tratare a întreruperilor vor începe la adresa 0, iată de ce secvența normală de program începe la adresa 02. De asemenea mai este de observat faptul că microprocesorul nu va răspunde la o nouă întrerupere atâta timp cât se află într-o subrutină de tratare a întreruperilor, prin aceasta se previne cazul în care întreruperile ar fi tot timpul sesizate, dar niciodată apelată subrutina de tratare.

Inserați codul în editorul VHDL, verificați sintaxa şi verificați dacă este sintetizabil.

4. Asamblarea proiectului

Creați un proiect nou, adăugați la proiect toate fişierele VHDL create începând cu laboratorul 8. Creați simboluri schematice corepunzătoare fiecărei componente VHDL. Adăugați proiectului un fişier schematic nou. Deschideți fişierul şi cu dublu click pe marginea dublă a foi de editare deschideți fereastra *Schematic Properties,*în care în câmpul *size* alegeți dimensiunea *A1 a* paginii de editare. Aduceți în pagină toate simbolurile corespunzătoare fişierelor VHDL adăugate proiectului şi aranjațile conform figurii 1 din cursul 10 (Etapele de proiectare în VHDL a unui microprocesor pe 8-biți). Conectați toate blocurile conform figurilor corespunzătoare din laboratoarele 8, 9, 10. Ca şi buffere tristate utilizați simbolurile standard din bibliteca schematic.

După ce s-au conectat toate simbolurile din fişierul schematic generați un fişier VHDL şi ataşați-l la prooeict. Verificați sintaxa fişierului VHDL generat şi faceți o sinteză a acestuia.