# Modélisation et implémentation d'un processeur RISC 16 Bits

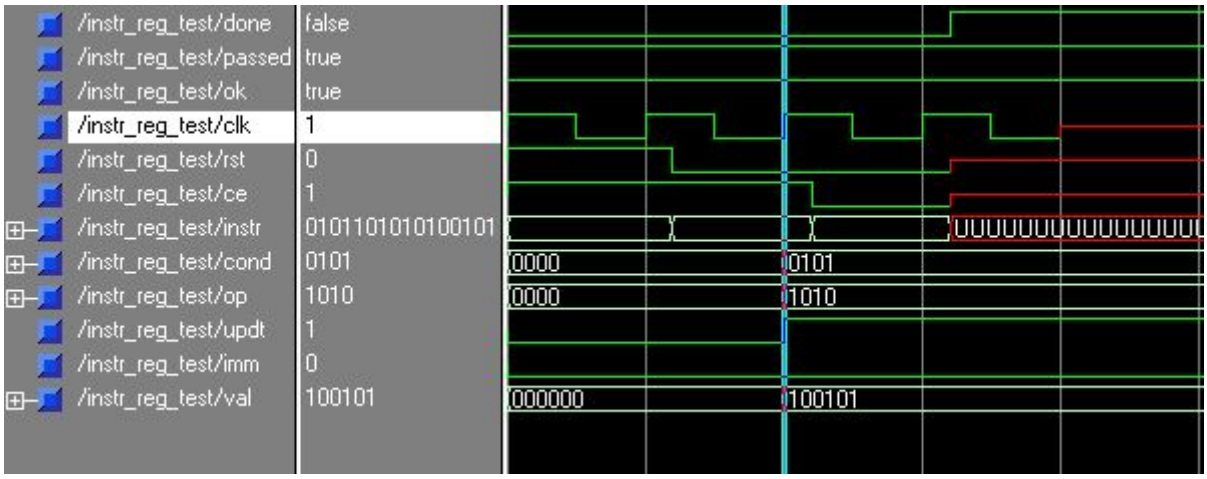
**Réalisé par:**
**Chaima Ghazouani et Emna Boussetta**
**2 ING ISEOC**

1. **Registre d'instruction**

- **Le programme**

```
26  use ieee.std_logic_1164.all;
27
28  entity instr_reg is
29    port ( clk : in  std_logic;
30           ce  : in  std_logic; --instr_ce
31           rst : in  std_logic; --reset
32
33           instr : in  std_logic_vector(15 downto 0); --ram_dout
34
35            --Mot d'instruction: cond.op.updt.imm.val
36           cond  : out std_logic_vector(3 downto 0);
37           op    : out std_logic_vector(3 downto 0);
38           updt  : out std_logic;
39           imm   : out std_logic;
40           val   : out std_logic_vector(5 downto 0) );
41
42  end instr_reg;
43
44  architecture arch of instr_reg is
45
46  begin
47  process(clk,rst,ce)
48  begin
49    if (rst ='1') then
50      cond<="0000";
51      op<="0000";
52      updt<='0';
53      imm<='0';
54      val<="000000";
55    elsif (clk'event and clk='1') and (ce='1') then
56
57        cond <= instr(15 downto 12);
58        op <= instr(11 downto 8);
59        updt <= instr(7);
60        imm <= instr(6);
61        val <= instr(5 downto 0);
62
63    end if;
64  end process ;
65
66  end arch;
67
```

- **La simulation**

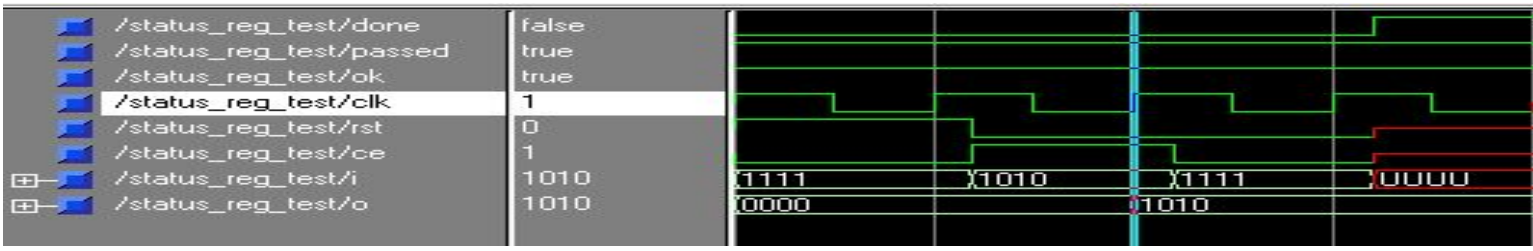



## 2. Le registre de statut

- **Le programme**

```
91   library ieee;
92   use ieee.std_logic_1164.all;
93
94
95   entity status_reg is
96     port ( clk : in  std_logic;
97            ce  : in  std_logic;--status_ce
98            rst : in  std_logic;--reset
99
100           i : in  std_logic_vector(3 downto 0);
101           o : out std_logic_vector(3 downto 0) );
102  end status_reg;
103
104  architecture arch of status_reg is
105  signal st : std_logic_vector(3 downto 0);
106  begin
107  process(clk ,rst,ce)
108  begin
109    if rst = '1' then
110       o<="0000";
111    end if ;
112    if clk'event and clk = '1' then
113      if (ce ='1') then
114        o<=i;
115      end if ;
116    end if ;
117  end process ;
118
119  end arch;
120
```

- **La simulation**



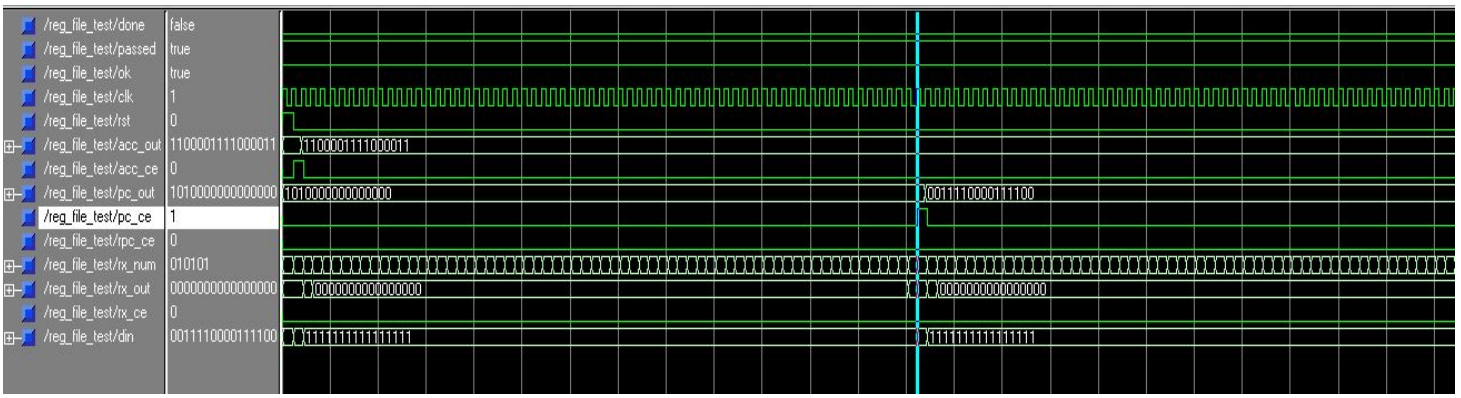
## 3. Banc de registre

### ❑ Le programme

```
155  library ieee;
156  use ieee.std_logic_1164.all;
157  use ieee.std_logic_unsigned.all;
158  use ieee.numeric_std.all;
159  entity reg_file is
160    port ( clk : in  std_logic;
161           rst : in  std_logic;
162           acc_out : out std_logic_vector(15 downto 0);
163           acc_ce  : in  std_logic;
164           pc_out : out std_logic_vector(15 downto 0);
165           pc_ce  : in  std_logic;
166           rpc_ce : in  std_logic;
167           rx_num : in  std_logic_vector(5 downto 0);
168           rx_out : out std_logic_vector(15 downto 0);
169           rx_ce  : in  std_logic;
170           din : in  std_logic_vector(15 downto 0) );
171  end reg_file;
```

```
173  architecture arch of reg_file is
174  type banc_reg is array (63 downto 0) of std_logic_vector (0 to 15);
175  signal reg : banc_reg;
176  signal i: integer;
177  signal s:std_logic_vector(5 downto 0):=(others=>'0');
178  begin
179  s<=rx_num;
180  process (clk,rst)
181  begin
182    if (rst='1') then
183         boucle:FOR  i in 0 to 62 loop
184         reg(i) <= "0000000000000000";
185         end loop boucle;
186        reg(63) <= "1010000000000000";
187    elsif (rising_edge(clk))then
188       if (rx_ce='1' )then
189
190        reg(to_integer(unsigned(s)))<=din;
191         end if;
192       if ( acc_ce='1')then
193            reg(0) <= din;
194      end if;
195       if (rpc_ce='1'  )then
196              reg(62)<= din;
197      end if;
198       if ( pc_ce='1'  )then
199             reg(63)<= din;
200       end if;
201  end if;
202
203  end process;
204  rx_out<=reg(to_integer(unsigned(s)));
205  acc_out<=reg(0);
206  pc_out<=reg(63);
207  end arch;
```

- **La simulation**




## 4. L'unité arithmétique et logique (ALU)

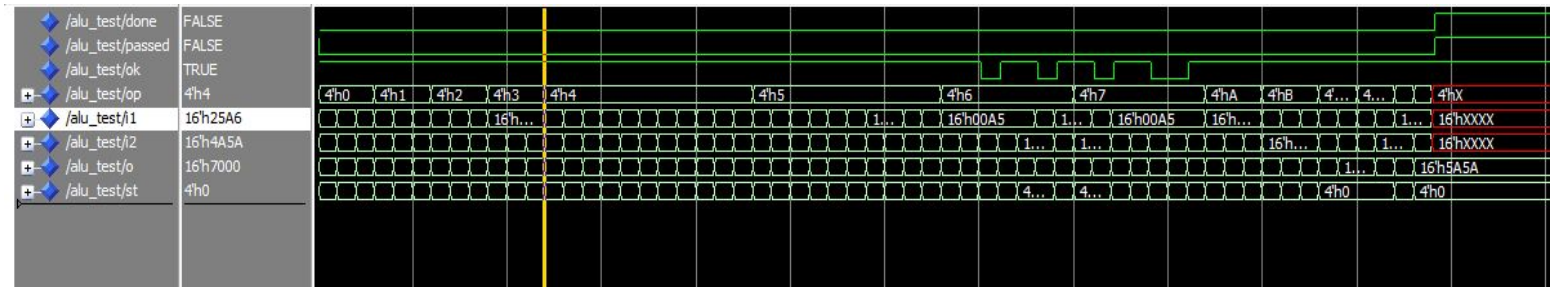
- **Le programme**

```
12  library IEEE;
13  use IEEE.STD_LOGIC_1164.ALL;
14  use ieee.std_logic_unsigned.all;
15  use ieee.numeric_std.all;
16
17  entity alu is
18    port ( op : in  std_logic_vector(3 downto 0);
19           i1 : in  std_logic_vector(15 downto 0);
20           i2 : in  std_logic_vector(15 downto 0);
21           o  : out std_logic_vector(15 downto 0);
22           st : out std_logic_vector(3 downto 0));
23  end alu;
24
25  architecture arch of alu is
26
27  signal result:std_logic_vector(16 downto 0);
28  signal Z,N,C,V:std_logic;
29  signal in1,in2:std_logic_vector(14 downto 0);
30  signal num,sgn1,sgn2:std_logic_vector(15 downto 0);
31
32  begin
```

```
31  begin
32  process(op,i1,i2)
33  begin
34  case op is
35  when "0000" =>result <='0'&i1 and '0'&i2; -- and
36  when "0001" =>result <='0'&i1 or '0'&i2; -- or
37  when "0010" =>result <='0'&i1 xor '0'&i2; -- xor
38  when "0011" =>result <='0'&(not i2) ; -- not
39  when "0100" =>result <=('0'&i1)+('0'&i2); -- add
40  when "0101" =>result <= std_logic_vector( signed('0'&i1(15 downto 0))- signed ('0'&i2(15 downto 0))); -- sub
41
42  -------------------
43  when "0110" =>result <=std_logic_vector(shift_left( signed('0'&i1),to_integer(signed('0'&i2))));  --lsl instruction
44  when "0111" => result<=std_logic_vector(shift_right( signed('0'&i1),to_integer(signed('0'&i2)))); --lsr instruction
45
46  -------------------
47  when "1000" => result <= ('0'&i2); --LDA instruction
48  when "1001" => result <= ('0'&i1); --STA instruction
49  --
50  when "1010" =>result <= '0'&i2;
51  when "1011" =>result <= '0'&i1;
52  when "1100" =>result(15 downto 0) <=i1+i2 ;
53  when "1101" =>result (15 downto 0)<=i1-i2;
54  when "1110" =>result (15 downto 0)<=i2;
55  when "1111" =>result (15 downto 0)<=i2;
56  when others => null;
57  end case;
58  end process;
59
```

```
54  o <= result(15 downto 0);
55  Z <='1' when (result(15 downto 0) ="0000000000000000") else '0'; --Zero
56  N <= '1' when(result(15)='1') else '0';
57  C <=  not result(16) when (op="0101") else
58        result(16);
59
60  in1 <= i1(14 downto 0);
61  in2 <= i2(14 downto 0);
62  num <=('0'&in1) + ('0'&in2);
63
64  V <= C xor num(15) when (op="0100")else
65       (i1(15)and (not i2(15)) and (not result(15))) or ((not i1(15))and i2(15)and result(15)) when (op="0101")
66      else
67      result(15) when (op="0110")else
68      '0';
69
70  st<= Z&N&C&V;
71  end arch;
72
```

### ❑ La simulation



## 5. Multiplexeur à deux entrée et une sortie

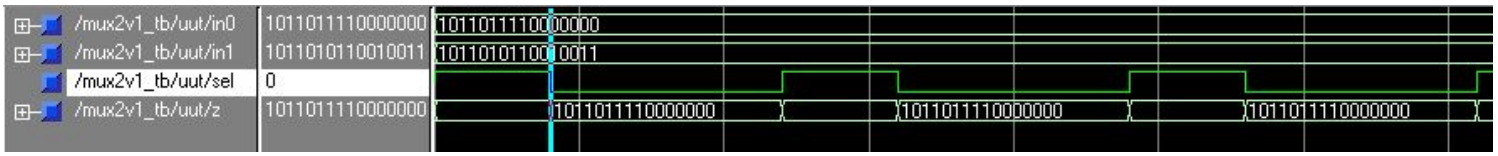
### ❑ Le programme

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2v1 is
        port (In0, In1: in std_logic_vector (15 downto 0);
     Sel: in std_logic;
                    Z: out std_logic_vector (15 downto 0));
end mux2v1;
architecture Arch of mux2v1 is
begin
Z <= In0 when Sel ='0' else
     In1 when Sel ='1' else
       "0000000000000000" ;
end Arch;

```

### ❑ Le test bench

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2v1_tb is
end mux2v1_tb;

architecture arch of mux2v1_tb is
component mux2v1
 port (
   In0: in std_logic_vector(15 downto 0);
   In1: in std_logic_vector(15 downto 0);
   Sel: in std_logic;
   Z: out std_logic_vector(15 downto 0));
end component;

signal tIn0,tIn1,tZ: std_logic_vector(15 downto 0);
signal tSel: std_logic;

begin
UUT: mux2v1 port map(tIn0,tIn1,tSel,tZ);
   tIn0<= "1011011110000000";
   tIn1 <= "1011010110010011";

process
begin
   tSel<= '1';
   wait for 20 ns;
   tSel<= '0';
   wait for 40 ns;
end process;

end arch;
```

❑ **La Simulation**




## 6. Multiplexeur à trois entrées et une sortie

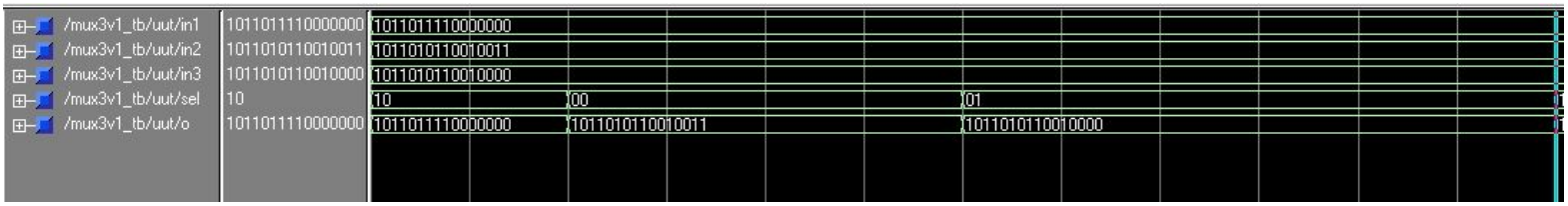
❑ **Le programme**

```
library ieee;
use ieee.std_logic_1164.all;

 entity mux3v1 is
      Port ( in1 : in  std_logic_vector (15 downto 0);
             in2 : in  std_logic_vector (15 downto 0);
             in3 : in  std_logic_vector (15 downto 0);
             sel : in std_logic_vector(1 downto 0);
            o : out std_logic_vector (15 downto 0));
 end mux3v1;


architecture arch of mux3v1 is
begin
o <= in1 when sel ="10"
   else in2 when sel ="00"
   else in3 when sel ="01"
   else "0000000000000000";
end arch;
```

- **test bench**

```
library ieee;
use ieee.std_logic_1164.all;

entity mux3v1_tb is
end mux3v1_tb;

architecture arch of mux3v1_tb is
component mux3v1
 port (
   in1: in std_logic_vector(15 downto 0);
   in2: in std_logic_vector(15 downto 0);
   in3: in std_logic_vector(15 downto 0);
   sel: in std_logic_vector(1 downto 0);
   o: out std_logic_vector(15 downto 0));
end component;

signal tIn1,tIn2,tIn3,too: std_logic_vector(15 downto 0);
signal tSel: std_logic_vector(1 downto 0);

begin
UUT: mux3v1 port map(tIn1,tIn2,tIn3,tSel,too);
   tIn1<= "1011011110000000";
   tIn2 <= "1011010110010011";
   tIn3 <= "1011010110010000";

process
begin
   tSel<= "10";
   wait for 20 ns;
   tSel<= "00";
   wait for 40 ns;
   tSel<="01";
   wait for 60 ns;
end process;

end arch;
```

- **La Simulation**



## 7. Incrementeur

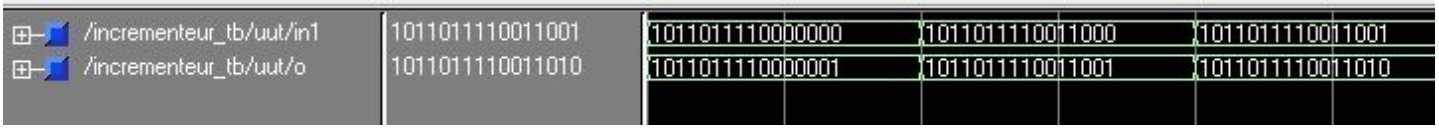
### ❑ Le programme

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity incrementeur is
   Port ( in1 : in  STD_LOGIC_VECTOR (15 downto 0);
                o : out STD_LOGIC_VECTOR (15 downto 0));
end incrementeur;

architecture Arch of incrementeur is

begin
o <= std_logic_vector(unsigned(in1)+ 1);
end Arch;

```

### ❑ test bench

```
library ieee;
use ieee.std_logic_1164.all;

entity incrementeur_tb is
end incrementeur_tb;

architecture arch of incrementeur_tb is

component incrementeur
   Port ( in1 : in  STD_LOGIC_VECTOR (15 downto 0);
                o : out STD_LOGIC_VECTOR (15 downto 0));
end component;

signal tin1, too: std_logic_vector(15 downto 0);


begin
UUT: incrementeur port map(tin1,too);
  tin1<= "1011011110000000","1011011110011000" after 20 ns,"1011011110011001" after 40 ns;


end arch;
```

### ❑ La Simulation



8. **Register OP1, OP2 ET res**

- **Le programme**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity reg is
port (clk :in std_logic ;
     rst:in std_logic ;
     i:in std_logic_vector(15 downto 0);
     ou :out std_logic_vector(15 downto 0));
end reg;

architecture arch of reg is
begin

process(clk,rst)
  begin
     if rst='1' then
        ou<= "0000000000000000";
     elsif (rising_edge(clk))then
        ou<=i;
     end if ;
  end process ;

end arch ;
```

- **test bench**

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_tb is
end reg_tb;

architecture arch of reg_tb is
   component reg
      port ( clk : in  std_logic;
             rst : in  std_logic;
            i : in std_logic_vector(15 downto 0);
             ou: out std_logic_vector(15 downto 0));
   end component;

signal ti,tou: std_logic_vector(15 downto 0);
signal clk,rst: std_logic;
begin
   UUT: reg port map(clk,rst,ti,tou);
   ti<= "1011011110000000";

rst<= '1','0' after 20 ns;
   process
   begin
      clk<= '0','1' after 10 ns;
   wait for 20 ns;
   end process;

end arch;
```

- **La Simulation**

## 9. L'unité de contrôle

### ❑ Le programme

```
59
60  architecture arch of control is
61    type state is (st_fetch1, st_fetch2, st_decode, st_exec, st_store);
62
63    signal state_0 : state;
64    signal state_r : state := st_fetch1;
65  begin
66    state_0 <= st_fetch2 when state_r = st_fetch1 else
67               st_decode when state_r = st_fetch2 else
68               st_exec   when state_r = st_decode else
69               st_store  when state_r = st_exec   else
70               st_fetch1 when state_r = st_store  else
71               state_r;
72
73   p1: process(state_r,instr_op,instr_cond,status,instr_updt )
74   begin
75
76
77           instr_ce<='0';
78           status_ce <='0';
79           acc_ce  <='0';
80           pc_ce   <='0';
81           rpc_ce <='0';
82           rx_ce  <='0';
83           ram_we <='0';
84            sel_ram_addr <='0';
85           sel_op1 <='0';
86           sel_rf_din <="00";
87
88   case state_r is
89
90    when st_fetch1 => sel_ram_addr <='0';
91
92    when st_fetch2 => instr_ce <='1';
93
```

```
 94      when st_decode =>
 95        if not ((instr_cond = "0001") --condition T
 96          or( instr_cond = "0010"and status="1000") --condition Z / status Z
 97          or( instr_cond = "0011"and status/="1000") --condition NZ / status not Z
 98          or( instr_cond = "0100"and not(status="1000"or status="0100")) --condition not(Z) et not(N)  status not(N ou Z)
 99          or( instr_cond = "0101"and (status="1000"or status="0100" or status="1100")) --condition Z ou N  status (Z ou N ou ZetN)
100          or( instr_cond = "0110"and status="0100") --condition N status N
101          or( instr_cond = "0111"and status/="0100") --condition not N status not N
102          or( instr_cond = "1000"and status="0010") --condition C status C
103          or( instr_cond = "1001"and status/="0010") --condition not C status not C
104          or( instr_cond = "1010"and status="0001") --condition V status V
105          or( instr_cond = "1011"and status/="0001")--condition not V status not V
106          or(instr_cond /= "0000" )) --condition not F
107        then           instr_ce    <= '0';
108                       status_ce   <= '0';
109                       acc_ce      <= '0';
110                       pc_ce       <= '0';
111                       rpc_ce      <= '0';
112                       rx_ce       <= '0';
113                       ram_we      <= '0';
114        elsif ( instr_op="0000" or instr_op="0001" or instr_op="0010" or --AND --OR --XOR --
115              instr_op="0100" or instr_op="0101" or instr_op="0110" or --ADD --SUB --LSL
116             instr_op="0111" or instr_op="1001" or instr_op="1011") --LSR --STR --MTR
117        then           instr_ce    <='0';
118                       status_ce   <='0';
119                       acc_ce      <='0';
120                       pc_ce       <='0';
121                       rpc_ce      <='0';
122                       rx_ce       <='0';
123                       ram_we      <='0';
124                       sel_op1     <='0';
125        elsif (instr_op="1100" or instr_op="1101") then --JRP --JRN
126                       instr_ce    <= '0';
127                       status_ce   <= '0';
128                       acc_ce      <= '0';
129                       pc_ce       <= '0';
130                       rpc_ce      <= '0';
131                       rx_ce       <= '0';
132                       ram_we      <= '0';
133                       sel_op1     <= '1';
```

```
134        else
135                       instr_ce    <= '0';
136                       status_ce   <= '0';
137                       acc_ce      <= '0' ;
138                       pc_ce       <= '0';
139                       rpc_ce      <= '0';
140                       rx_ce       <= '0';
141                       ram_we      <= '0';
142        end if;
143
144    when st_exec =>
145                    if not ((instr_cond = "0001") --condition T
146        or( instr_cond = "0010"and status="1000") --condition Z / status Z
147        or( instr_cond = "0011"and status/="1000") --condition NZ / status not Z
148        or( instr_cond = "0100"and not(status="1000" or status="0100")) --condition not(Z) et not(N)  status not(N ou Z)
149        or( instr_cond = "0101"and (status="1000"or status="0100" or status="1100")) --condition Z ou N  status (Z ou N ou ZetN)
150        or( instr_cond = "0110"and status="0100") --condition N status N
151        or( instr_cond = "0111"and status/="0100") --condition not N status not N
152        or( instr_cond = "1000"and status="0010") --condition C status C
153        or( instr_cond = "1001"and status/="0010") --condition not C status not C
154        or( instr_cond = "1010"and status="0001") --condition V status V
155        or( instr_cond = "1011"and status/="0001")) --condition not V status not V
156
157        then
158                       instr_ce    <= '0';
159                       status_ce   <= '0';
160                       acc_ce      <= '0';
161                       pc_ce       <= '1';
162                       rpc_ce      <= '0';
163                       rx_ce       <= '0';
164                       ram_we      <= '0';
165                       sel_rf_din  <= "10" ;
```

```
126         elsif (instr_op="1100" or instr_op="1101") then --JRP --JRN
127                         instr_ce    <= '0';
128                         status_ce   <= '0';
129                         acc_ce      <= '0';
130                         pc_ce       <= '0';
131                         rpc_ce      <= '0';
132                         rx_ce       <= '0';
133                         ram_we      <= '0';
134                         sel_op1     <= '1';
135         else
136                         instr_ce    <= '0';
137                         status_ce   <= '0';
138                         acc_ce      <= '0' ;
139                         pc_ce       <= '0';
140                         rpc_ce      <= '0';
141                         rx_ce       <= '0';
142                         ram_we      <= '0';
143         end if;
144
145     when st_exec =>
146                     if not ((instr_cond = "0001") --condition T
147         or( instr_cond = "0010"and status="1000") --condition Z / status Z
148         or( instr_cond = "0011"and status/="1000") --condition NZ / status not Z
149         or( instr_cond = "0100"and not(status="1000" or status="0100")) --condition not(Z) et not(N)  status not(N ou Z)
150         or( instr_cond = "0101"and (status="1000"or status="0100" or status="1100")) --condition Z ou N  status (Z ou N ou ZetN)
151         or( instr_cond = "0110"and status="0100") --condition N status N
152         or( instr_cond = "0111"and status/="0100") --condition not N status not N
153         or( instr_cond = "1000"and status="0010") --condition C status C
154         or( instr_cond = "1001"and status/="0010") --condition not C status not C
155         or( instr_cond = "1010"and status="0001") --condition V status V
156         or( instr_cond = "1011"and status/="0001")) --condition not V status not V
157
158         then
159                         instr_ce    <= '0';
160                         status_ce   <= '0';
161                         acc_ce      <= '0';
162                         pc_ce       <= '1';
163                         rpc_ce      <= '0';
164                         rx_ce       <= '0';
165                         ram_we      <= '0';
166                         sel_rf_din  <= "10" ;
167
```

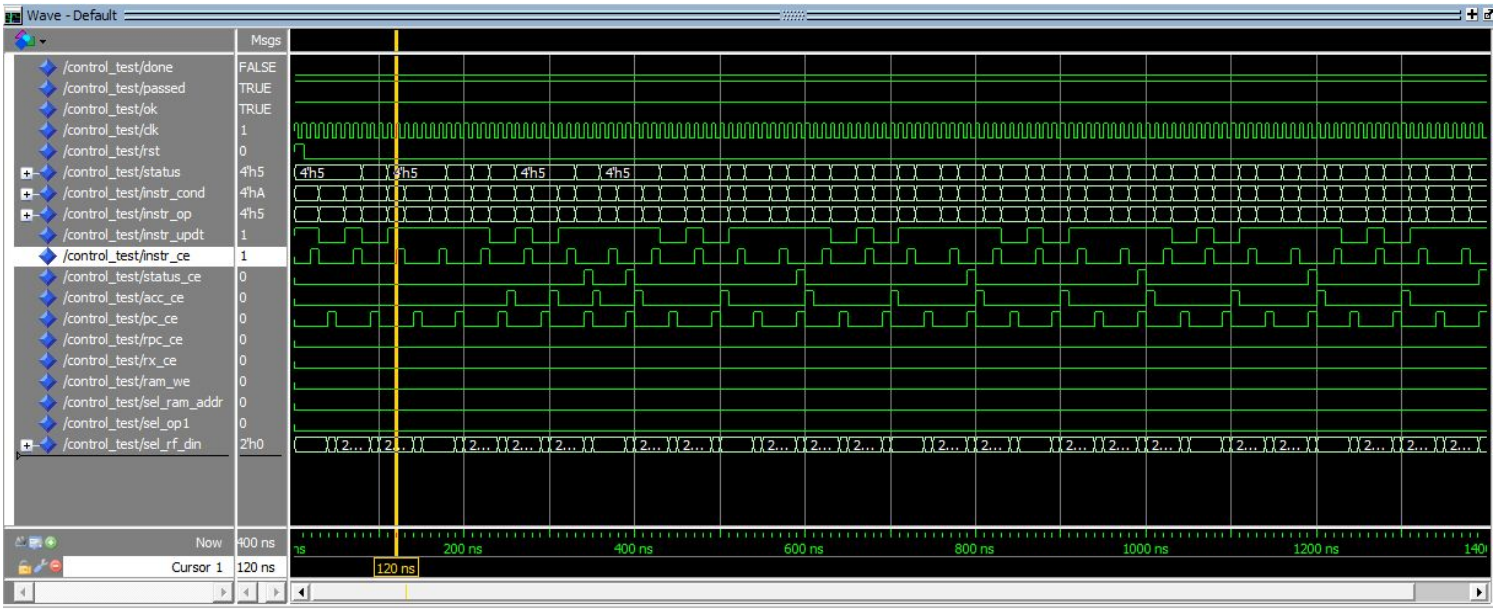
```
167
168          elsif (instr_op="1000" ) then --LDA
169                         instr_ce    <= '0';
170                         status_ce   <= '0';
171                         acc_ce      <= '0';
172                         pc_ce       <= '1';
173                         rpc_ce      <= '0';
174                         rx_ce       <= '0';
175                         ram_we      <= '0';
176                         sel_ram_addr<= '1';
177                         sel_rf_din  <= "10";
178
179          elsif (instr_op="1001" ) then --STA
180                         instr_ce    <= '0';
181                         status_ce   <= '0';
182                         acc_ce      <= '0';
183                         pc_ce       <= '1';
184                         rpc_ce      <= '0';
185                         rx_ce       <= '0';
186                         ram_we      <= '1';
187                         sel_ram_addr<= '1';
188                         sel_rf_din  <= "10";
189          elsif (instr_op="1111" ) then --CAL
190                         instr_ce    <= '0';
191                         status_ce   <= '0';
192                         acc_ce      <= '0';
193                         rpc_ce      <= '1';
194                         rx_ce       <= '0';
195                         ram_we      <= '0';
196                         sel_rf_din  <= "10";
197          elsif (instr_op="1100" or instr_op="1101" or instr_op="1110") then --JRP --JRN --JPR
198                         instr_ce    <= '0';
199                         status_ce   <= '0';
200                         acc_ce      <= '0';
201                         rpc_ce      <= '0';
202                         rx_ce       <= '0';
203                         ram_we      <= '0';
```

```
204
205         elsif( instr_updt='0' )  then
206                        instr_ce    <= '0';
207                         status_ce   <= '0';
208                         acc_ce      <= '0';
209                         pc_ce       <= '1';
210                         rpc_ce      <= '0';
211                         rx_ce       <= '0';
212                         ram_we      <= '0';
213                         sel_rf_din  <= "10";
214         else
215
216                         instr_ce    <= '0' ;
217                         status_ce   <= '1' ;
218                         acc_ce      <= '0' ;
219                         pc_ce       <= '1' ;
220                         rpc_ce      <= '0' ;
221                         rx_ce       <= '0' ;
222                         ram_we      <= '0' ;
223                         sel_rf_din  <= "10";
224         end if;
225     when st_store =>
226         if not((instr_cond = "0001") --condition T
227         or( instr_cond = "0010"and status="1000") --
228         or( instr_cond = "0011"and status/="1000")
229         or( instr_cond = "0100"and not(status="1000"or status="0100"))
230         or( instr_cond = "0101"and (status="1000"or status="0100" or status="1100"))
231         or( instr_cond = "0110"and status="0100")
232         or( instr_cond = "0111"and status/="0100")
233         or( instr_cond = "1000"and status="0010")
234         or( instr_cond = "1001"and status/="0010")
235         or( instr_cond = "1010"and status="0001")
236         or( instr_cond = "1011"and status/="0001") )
237         then
238                        instr_ce    <= '0';
239                         status_ce   <= '0';
240                         acc_ce      <= '0';
241                         pc_ce       <= '0';
242                         rpc_ce      <= '0';
243                         rx_ce       <= '0';
244                         ram_we      <= '0';
```

```
245           elsif (instr_op="1100" or instr_op="1101" or instr_op="1110" or instr_op="1111" ) then
246          instr_ce    <= '0';
247                     status_ce   <= '0';
248                     acc_ce      <= '0';
249                     pc_ce       <= '1';
250                     rpc_ce      <= '0';
251                     rx_ce       <= '0';
252                     ram_we      <= '0';
253                     sel_rf_din  <= "00";
254         elsif (instr_op="1011") then
255          instr_ce    <= '0';
256                     status_ce   <= '0';
257                     acc_ce      <= '0';
258                     pc_ce       <= '0';
259                     rpc_ce      <= '0';
260                     rx_ce       <= '1';
261                     ram_we      <= '0';
262                     sel_rf_din  <= "00";
263         elsif (instr_op="1001") then
264          instr_ce    <= '0';
265          status_ce   <= '0';
266          acc_ce      <= '0';
267                     pc_ce       <= '0';
268                     rpc_ce      <= '0';
269                     rx_ce       <= '0';
270                     ram_we      <= '0';
271         elsif (instr_op="1000") then
272          instr_ce    <= '0';
273                     status_ce   <= '0';
274                     acc_ce      <= '1';
275                     pc_ce       <= '0';
276                     rpc_ce      <= '0';
277                     rx_ce       <= '0';
278                     ram_we      <= '0';
279                     sel_rf_din  <= "01";
```

```
280         else
281             instr_ce     <= '0';
282                         status_ce    <= '0';
283                         acc_ce       <= '1';
284                         pc_ce        <= '0';
285                         rpc_ce       <= '0';
286                         rx_ce        <= '0';
287                         ram_we       <= '0';
288                         sel_rf_din   <= "00";
289
290         end if;
291     end case;
292     end process;
293
294
295     p2: process(clk, rst)
296       begin
297         if rst = '1' then
298           state_r <= st_fetch1;
299         elsif clk'event and clk = '1' then
300           state_r <= state_0;
301         end if;
302       end process p2;
303
304     end arch;
```

❑ **La simulation**



## 10. processeur RISC

### ❑ Le programme

```
14  library ieee;
15  use ieee.std_logic_1164.all;
16  use ieee.std_logic_unsigned.all;
17
18  entity proc is
19    port ( clk : in  std_logic;
20           rst : in  std_logic;
21
22           ram_addr : out std_logic_vector(15 downto 0);
23           ram_din  : out std_logic_vector(15 downto 0);
24           ram_dout : in  std_logic_vector(15 downto 0);
25           ram_we   : out std_logic );
26  end proc;
27
28  architecture arch of proc is
29
30    component control
31      port ( clk : in  std_logic;
32             rst : in  std_logic;
33
34             status     : in  std_logic_vector(3 downto 0);
35             instr_cond : in  std_logic_vector(3 downto 0);
36             instr_op   : in  std_logic_vector(3 downto 0);
37             instr_updt : in  std_logic;
38
39             instr_ce  : out std_logic;
40             status_ce : out std_logic;
41             acc_ce    : out std_logic;
42             pc_ce     : out std_logic;
43             rpc_ce    : out std_logic;
44             rx_ce     : out std_logic;
45
46             ram_we : out std_logic;
47
48             sel_ram_addr : out std_logic;
49             sel_op1      : out std_logic;
50             sel_rf_din   : out std_logic_vector(1 downto 0) );
51    end component;
52
```

```
53
54      component alu
55
56        port ( op : in  std_logic_vector(3 downto 0);
57               i1 : in  std_logic_vector(15 downto 0);
58               i2 : in  std_logic_vector(15 downto 0);
59               o  : out std_logic_vector(15 downto 0);
60               st : out std_logic_vector(3 downto 0) );
61      end component;
62
63
64
65       component status_reg
66       port ( clk : in  std_logic;
67              ce  : in  std_logic;
68              rst : in  std_logic;
69
70              i : in  std_logic_vector(3 downto 0);
71              o : out std_logic_vector(3 downto 0) );
72      end component;
73
74
75        component instr_reg
76       port ( clk : in  std_logic;
77              ce  : in  std_logic;
78              rst : in  std_logic;
79
80              instr : in  std_logic_vector(15 downto 0);
81              cond  : out std_logic_vector(3 downto 0);
82              op    : out std_logic_vector(3 downto 0);
83              updt  : out std_logic;
84              imm   : out std_logic;
85              val   : out std_logic_vector(5 downto 0) );
86      end component;
87
```

```
88
89      component reg_file
90        port ( clk : in  std_logic;
91               rst : in  std_logic;
92
93               acc_out : out std_logic_vector(15 downto 0);
94               acc_ce  : in  std_logic;
95
96               pc_out : out std_logic_vector(15 downto 0);
97               pc_ce  : in  std_logic;
98               rpc_ce : in  std_logic;
99
100              rx_num : in  std_logic_vector(5 downto 0);
101              rx_out : out std_logic_vector(15 downto 0);
102              rx_ce  : in  std_logic;
103
104              din : in  std_logic_vector(15 downto 0) );
105     end component;
106
107
108
109     component mux2v1
110             port (In0, In1: in std_logic_vector (15 downto 0);
111          Sel: in std_logic;
112                         Z: out std_logic_vector (15 downto 0));
113     end component;
114
115
116      component mux3v1 -------
117           Port ( in1 : in  std_logic_vector (15 downto 0);
118                  in2 : in  std_logic_vector (15 downto 0);
119                  in3 : in  std_logic_vector (15 downto 0);
120                  sel :in std_logic_vector(1 downto 0);
121                 o : out std_logic_vector (15 downto 0));
122      end component;
123
```

```
124
125      component incrementeur -------
126           Port ( in1 : in  STD_LOGIC_VECTOR (15 downto 0);
127                  o : out STD_LOGIC_VECTOR (15 downto 0));
128      end component;
129
130     component reg
131     port (clk :in std_logic ;
132          rst:in std_logic ;
133          i:in std_logic_vector(15 downto 0);
134          ou :out std_logic_vector(15 downto 0));
135     end component;
136
137     signal stat,icond,iop,oia :std_logic_vector(3 downto 0);
138     signal iupdt,iimm,i_ce,stat_ce,a_ce,p_ce,rp_ce,r_ce,sel_ram,sel_op :std_logic;
139     signal sel_rf :std_logic_vector(1 downto 0);
140     signal op1,op2,res,acc,pc,rx,dreg,o1,o2,incresult,vall2,inc :std_logic_vector(15 downto 0);
141     signal vall:std_logic_vector(5 downto 0);
142
```

```
143  begin
144
145  CU: control port map (clk,rst,stat,icond,iop,iupdt,i_ce,stat_ce,a_ce,p_ce,rp_ce,r_ce,ram_we,sel_ram,sel_op,sel_rf);
146  --clk, rst, status ,instr_cond, instr_op, instr_updt ,instr_ce, status_ce, acc_ce , pc_ce ,rpc_ce,rx_ce ,ram_we,sel_ram_addr,sel_op1 ,sel_rf_din
147
148  RS: status_reg port map(clk,stat_ce,rst,oia,stat);
149  -- clk ,ce ,rst,i,o
150
151
152  RI:instr_reg port map(clk,i_ce,rst,ram_dout,icond,iop,iupdt,iimm,vall);
153  ---clk , ce ,rst,instr ,cond ,op ,updt ,imm ,val
154
155  BR:reg_file port map(clk,rst,acc,a_ce,pc,p_ce,rp_ce,vall,rx,r_ce,dreg);
156  --clk,rst,acc_out ,acc_ce ,pc_out ,pc_ce ,  rpc_ce , rx_num ,rx_out ,rx_ce ,din
157
158  vall2<= "0000000000"&vall ;
159  M1:mux2v1 port map (rx,vall2,iimm,o2); -----
160  --IN0,In1,Sel,Z
161
162  OP2R: reg port map (clk,rst,o2,op2); ------
163  --clk , rst, i,ou
164
165  M2:mux2v1 port map (acc,pc,sel_op,o1);--o1=>ram_din); ------
166  --IN0,In1,Sel,Z
167
168  OP1R: reg port map (clk,rst,o1,op1); -----
169  --clk ,rst,i,ou
170
171  AALU: alu port map (iop,op1,op2,res,oia);
172  --op ,i1,i2 , o ,st
173
174  INCR: incrementeur port map (pc,inc); -----
175
176  RRES: reg port map (clk,rst,res,incresult); -------
177
178  M3:mux3v1 port map (inc,incresult,ram_dout,sel_rf,dreg); ------
179  --in1 ,in2,in3 ,sel ,o
180
181  M4:mux2v1 port map(pc,op2,sel_ram,ram_addr);
182
```

### ❑ La simulation