```
1.
2.  LIBRARY IEEE;
3.  USE IEEE.STD_LOGIC_1164.ALL;
4.  USE IEEE.STD_LOGIC_ARITH.ALL;
5.  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
6.
7.  ENTITY my_scomp_v0_0 IS
8.  PORT( reloj : IN STD_LOGIC;
9.                  reset : IN STD_LOGIC;
10.                 IR_out: out std_logic_vector(15 downto 0);
11.                 AC_out: out std_logic_vector(15 downto 0);
12.                 PC_out : out std_logic_vector(7 downto 0);
13.                 IO_input : in  std_logic_vector(7 DOWNTO 0);
14.                 IO_output : out std_logic_vector(7 DOWNTO 0)
15.                 );
16. END my_scomp_v0_0;
17.
18. ARCHITECTURE rtl OF my_scomp_v0_0 IS
19.
20.         SIGNAL MEMq, MEMdata: STD_LOGIC_VECTOR(15 DOWNTO 0 );
21.         SIGNAL MEMadr : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
22.         SIGNAL MEMwe : STD_LOGIC;
23.
24.         -- Declaracion del IP core ram_256_x_16
25.         --
26.         --      Memoria RAM de un puerto, 256 palabras, 16 bits por palabra,
27.         --      Entradas de datos, direcciones y control de memoria REGISTRADAS,
28.         --      Salida NO REGISTRADA
29.         --      Fichero de inicializacion: programa.mif
30.
31.
32.         component IP_ram_256_x_16
33.         PORT
34.         (
35.                 address         : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
36.                 clock           : IN STD_LOGIC  := '1';
37.                 data            : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
38.                 wren            : IN STD_LOGIC ;
39.                 q               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
40.         );
41.         end component;
42.
43.
44.         -- Declaracion del componente con versión inicial del procesador
45.         --
46.
47.         COMPONENT procesador_v2_3 is
48.         PORT( clock : IN STD_LOGIC;
49.                 reset : IN STD_LOGIC;
50.                 AC_out : out std_logic_vector(15 downto 0);
51.                 IR_out : out std_logic_vector(15 downto 0);
52.                 PC_out : out std_logic_vector(7 downto 0);
53.                 MEMq : in std_logic_vector(15 downto 0);
54.                 MEMdata: out std_logic_vector(15 downto 0);
55.                 MEMwe : out std_logic;
56.                 MEMadr : out std_logic_vector(7 downto 0);
57.                 IO_input : in  std_logic_vector(7 DOWNTO 0);
58.                 IO_output : out std_logic_vector(7 DOWNTO 0)
59.         );
60.         END COMPONENT;
61.
62.         BEGIN
63.
64.         -- Instancia denominada MEM del IP core ram_256_x_16
65.         --
66.
67.         MEM: IP_ram_256_x_16 PORT MAP (
68.                 address   => MEMadr,
69.                 clock     => reloj,
70.                 data      => MEMdata,
71.                 wren      => MEMwe,
72.                 q         => MEMq
73.         );
74.
75.
76.         -- Instancia denominada PROC de la version inicial del procesador
```

```
77.
78.          PROC: procesador_v2_3 PORT MAP (
79.                   clock => reloj,
80.                   reset => reset,
81.                   AC_out => AC_out,
82.                   IR_out => IR_out,
83.                   PC_out => PC_out,
84.                   MEMq => MEMq,
85.                   MEMdata => MEMdata,
86.                   MEMwe => MEMwe,
87.                   MEMadr => MEMadr,
88.                   IO_input => IO_input,
89.                   IO_output => IO_output
90.          );
91.
92.
93.
94. END rtl;
```

```
1.  -- Descripción de una procesador que ejecuta cuatro instrucciones.
2.  -- Basado en ejemplo de Hamblen, J.O., Hall T.S., Furman, M.D.:
3.  -- Rapid Prototyping of Digital Systems : SOPC Edition, Springer 2008.
4.  -- (Capítulo 9)
5.
6.
7.  LIBRARY IEEE;
8.  USE IEEE.STD_LOGIC_1164.ALL;
9.  USE IEEE.STD_LOGIC_ARITH.ALL;
10. USE IEEE.STD_LOGIC_UNSIGNED.ALL;
11.
12. ENTITY procesador_v2_3 IS
13. PORT( clock : IN STD_LOGIC;
14.                 reset : IN STD_LOGIC;
15.                 AC_out : out std_logic_vector(15 downto 0);
16.                 IR_out : out std_logic_vector(15 downto 0);
17.                 PC_out : out std_logic_vector(7 downto 0);
18.                 MEMq : in std_logic_vector(15 downto 0);
19.                 MEMdata: out std_logic_vector(15 downto 0);
20.                 MEMwe : out std_logic;
21.                 MEMadr : out std_logic_vector(7 downto 0);
22.                 IO_input : in  std_logic_vector(7 DOWNTO 0);
23.                 IO_output : out std_logic_vector(7 DOWNTO 0)
24.                 );
25. END procesador_v2_3;
26.
27. ARCHITECTURE rtl OF procesador_v2_3 IS
28.         TYPE STATE_TYPE IS ( reset_pc, fetch1, decode, add1, add2, sub1, sub2,
29.                                                                 nand1, nand2, jneg, jpos,
    jzero, load1, store0, store1,
30.                                                                 shiftl, shiftr, ins_in,
    ins_out, jump);
31.         SIGNAL state: STATE_TYPE;
32.         SIGNAL IR, AC, TEMP: STD_LOGIC_VECTOR(15 DOWNTO 0 );
33.         SIGNAL IO_IN, IO_OUT: STD_LOGIC_VECTOR(7 DOWNTO 0);
34.         SIGNAL PC : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
35.
36.         BEGIN
37.
38.
39.         -- Asignaciones a puertos de salida
40.         --
41.         AC_out <= AC;
42.         IR_out <= IR;
43.         PC_out <= PC;
44.         IO_output <= IO_OUT;
45.         IO_IN <= IO_input;
46.
47.
48. FSMD: PROCESS ( CLOCK, RESET, state, PC, AC, IR, TEMP, IO_IN, IO_OUT )
49.
50. BEGIN
51.
52. -- Asignaciones a REGISTROS en datapath y MAQUINA DE ESTADOS de la unidad de control
53. IF reset = '1' THEN
54.         state <= reset_pc;
55.         ELSIF clock'EVENT AND clock = '1' THEN
56.          CASE state IS
57.                 WHEN reset_pc =>
58.                         PC       <= "00000000";
59.                         AC <= "0000000000000000";
60.                         state <= fetch1;
61.                 WHEN fetch1 =>
62.                         IR <= MEMq;
63.                         PC <= PC + 1;
64.                         state <= decode;
65.                 WHEN decode =>
66.                         CASE IR( 15 DOWNTO 8 ) IS
67.                                 WHEN "00000000" =>
68.                                         state <= add1;
69.                                 WHEN "00000001" =>
70.                                         state <= store0;
71.                                 WHEN "00000010" =>
72.                                         state <= load1;
73.                                 WHEN "00000011" =>
74.                                         state <= jump;
```

```
75.                                 WHEN "00000100" =>
76.                                         state <= sub1;
77.                                 WHEN "00000101" =>
78.                                         state <= nand1;
79.                                 WHEN "00000110" =>
80.                                         state <= jneg;
81.                                 WHEN "00000111" =>
82.                                         state <= jpos;
83.                                 WHEN "00001000" =>
84.                                         state <= jzero;
85.                                 WHEN "00001001" =>
86.                                         state <= shiftl;
87.                                 WHEN "00001010" =>
88.                                         state <= shiftr;
89.                                 WHEN "00001011" =>
90.                                         state <= ins_in;
91.                                 WHEN "00001100" =>
92.                                         state <= ins_out;
93.                                 WHEN OTHERS =>
94.                                         state <= fetch1;
95.                         END CASE;
96.                 WHEN add1 =>
97.                         TEMP <= MEMq;
98.                         state <= add2;
99.                 WHEN add2 =>
100.                        AC <= AC + TEMP;
101.                        state <= fetch1;
102.                WHEN store0 =>
103.                        state <= store1;
104.                WHEN store1 =>
105.                        state <= fetch1;
106.                WHEN load1 =>
107.                        AC <= MEMq;
108.                        state <= fetch1;
109.                WHEN jump =>
110.                        PC <= IR( 7 DOWNTO 0 );
111.                        state <= fetch1;
112.                WHEN jneg =>
113.                        IF AC(15) = '1' THEN
114.                                PC <= IR(7 DOWNTO 0);
115.                        END IF;
116.                        state <= fetch1;
117.                WHEN jpos =>
118.                        IF AC(15) = '0' AND AC > 0 THEN
119.                                PC <= IR(7 DOWNTO 0);
120.                        END IF;
121.                        state <= fetch1;
122.                WHEN jzero =>
123.                        IF AC = "0000000000000000" THEN
124.                                PC <= IR(7 DOWNTO 0);
125.                        END IF;
126.                        state <= fetch1;
127.                WHEN sub1 =>
128.                        TEMP <= MEMq;
129.                        state <= sub2;
130.                WHEN sub2 =>
131.                        AC <= AC - TEMP;
132.                        state <= fetch1;
133.                WHEN nand1 =>
134.                        TEMP <= MEMq;
135.                        state <= nand2;
136.                WHEN nand2 =>
137.                        AC <= AC NAND TEMP;
138.                        state <= fetch1;
139.                WHEN shiftl =>
140.                        AC <= SHL(AC, IR(7 DOWNTO 0));
141.                        state <= fetch1;
142.                WHEN shiftr =>
143.                        AC <= SHR(AC, IR(7 DOWNTO 0));
144.                        state <= fetch1;
145.                WHEN ins_in =>
146.                        IF IR(0) = '0' THEN
147.                                AC(7 DOWNTO 0) <= IO_IN;
148.                        ELSE
149.                                AC(15 DOWNTO 8) <= IO_IN;
150.                        END IF;
```

```
151.                            state <= fetch1;
152.                    WHEN ins_out =>
153.                            IF IR(0) = '0' THEN
154.                                    IO_OUT <= AC(7 DOWNTO 0);
155.                            ELSE
156.                                    IO_OUT <= AC(15 DOWNTO 8);
157.                            END IF;
158.                            state <= fetch1;
159.                    WHEN OTHERS =>
160.                            state <= fetch1;
161.            END CASE;
162.           END IF;
163.
164. -- Asignaciones a BUSES de entrada a MEMORIA (Direcciones, Datos y control de escritura)
165.
166.           CASE state IS
167.                    WHEN reset_pc =>
168.                            MEMadr <= "00000000";
169.                            MEMwe <= '0';
170.                            MEMdata <= (others =>'-');
171.                    WHEN ins_in | ins_out | shiftl | shiftr | add2 | store1 | load1 | sub2 | nand2 =>
172.                            MEMadr <= PC;
173.                            MEMwe <= '0';
174.                            MEMdata <= (others =>'-');
175.                    WHEN jneg =>
176.                            IF AC(15) = '1' THEN
177.                                    MEMadr <= IR(7 downto 0);
178.                                    MEMwe <= '0';
179.                                    MEMdata <= (others => '-');
180.                            ELSE
181.                                    MEMadr <= PC;
182.                                    MEMwe <= '0';
183.                                    MEMdata <= (others =>'-');
184.                            END IF;
185.                    WHEN jpos =>
186.                            IF AC(15) = '0' AND AC > 0 THEN
187.                                    MEMadr <= IR(7 downto 0);
188.                                    MEMwe <= '0';
189.                                    MEMdata <= (others => '-');
190.                            ELSE
191.                                    MEMadr <= PC;
192.                                    MEMwe <= '0';
193.                                    MEMdata <= (others =>'-');
194.                            END IF;
195.                    WHEN jzero =>
196.                            IF AC = "0000000000000000" THEN
197.                                    MEMadr <= IR(7 downto 0);
198.                                    MEMwe <= '0';
199.                                    MEMdata <= (others => '-');
200.                            ELSE
201.                                    MEMadr <= PC;
202.                                    MEMwe <= '0';
203.                                    MEMdata <= (others =>'-');
204.                            END IF;
205.                    WHEN store0  =>
206.                            MEMadr <= IR(7 downto 0);
207.                            MEMwe <= '1';
208.                            MEMdata <= AC;
209.                    WHEN others => -- fetch, decode, add1, sub1, nand1, jump
210.                            MEMadr <= IR(7 downto 0);
211.                            MEMwe <= '0';
212.                            MEMdata <= (others =>'-');
213.           end case;
214.
215. END PROCESS;
216.
217.
218. END rtl;
```