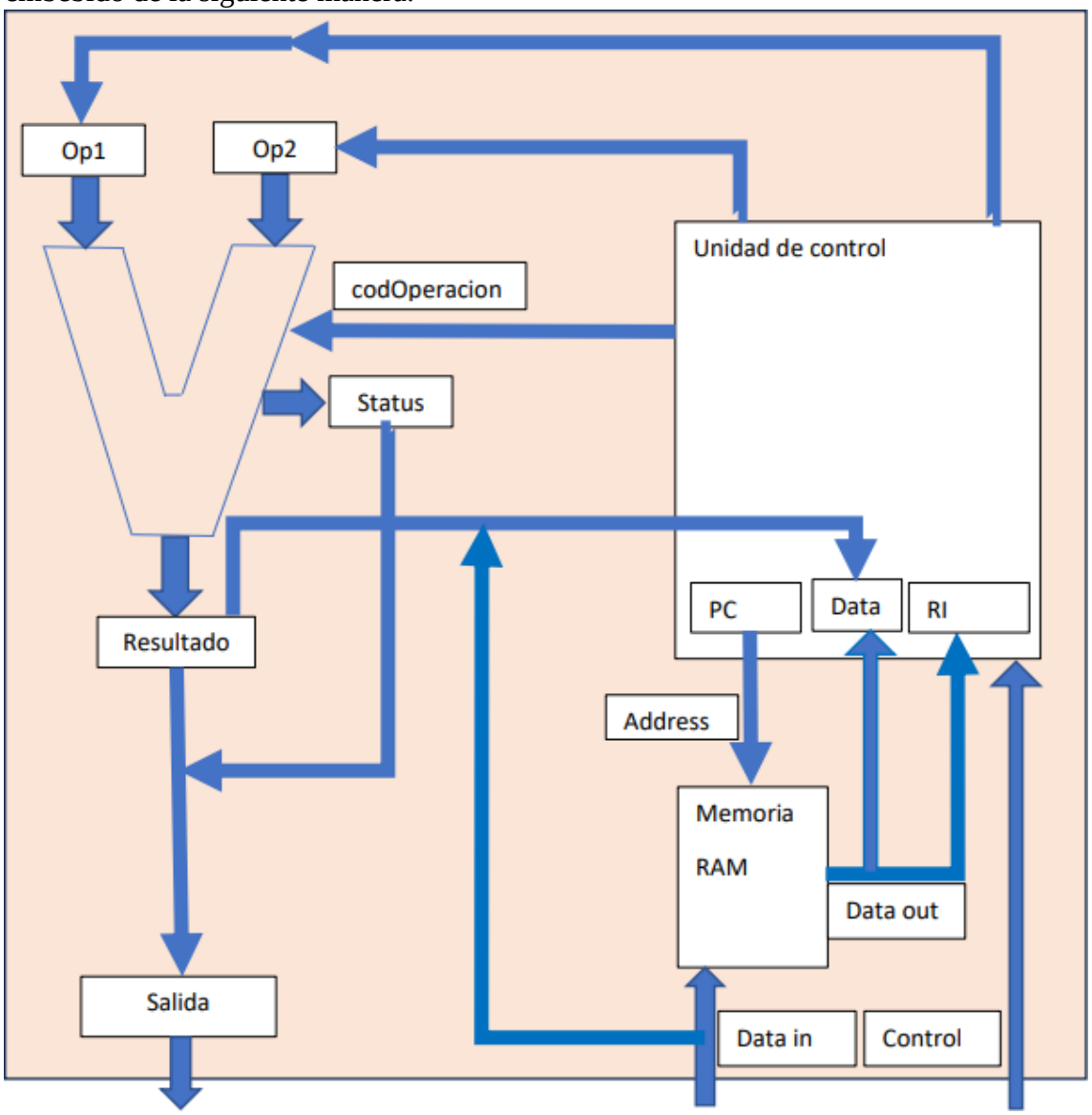
Practico Diseño de Sistemas Digitales Parte 3

Alumnos: Chamorro, Juan – Valori, Tomás

Facultad de Ciencia y Tecnología, Universidad Autónoma de Entre Ríos

Asignatura: Diseño de Sistemas Digitales
Profesor: Eduardo Velazquez, Gisela Giménez
Fecha de vencimiento:

La misión del trabajo fue realizar, en descripción VHDL, un sistema microprocesado embebido de la siguiente manera:



En el cual se pedían los siguientes registros de 8 bits: Op1 – Op2 – Resultado – Status – Salida - PC (contador de programa) – RI (registro de instrucción) – Data (registro de datos) – Memoria RAM de 64 celdas y 8 bits por celda – ALU de 8 bits con 7 operaciones (suma, resta AND, OR, SHL, SHR, NOT), con salida de resultado y registro de status con banderas resultantes de la operación realizada (Cy, OV, Z).

Dichos elementos deberían ser implementados como componentes al sistema general.

El ciclo de instrucción era de la siguiente manera:

- Leer instrucción y cargarla en RI.
- Decodificar la instrucción presente en RI.
- Ejecutar instrucción.
- Incrementar PC + 1.

Se piden 15 instrucciones específicas que veremos punto por punto en la realización del proyecto.

La señal de control está compuesta por las señales de clock (1 bit), load (1 bit) y reset (1 bit).

Cuando reset está en cero el PC se coloca a cero.

Cuando Load está en cero, se ejecuta el programa en forma normal.

Cuando Load está en uno, por cada clock de reloj se carga un valor en la memoria por la entrada Data_in.

Por último, para corroborar que todo funciona como corresponde, se pide realizar un testbench, por medio de un programa que utilice todas las instrucciones.

REALIZACION DEL PROYECTO

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
```

**Estas fueron las librerías que utilizamos.*

```
6  entity cpu_nuevo is
7      port(
8          data_in: in std_logic_vector (7 downto 0);
9          control: in std_logic_vector (2 downto 0);
10         salida: out std_logic_vector (7 downto 0));
11 end entity cpu_nuevo;
```

**Entidad con sus respectivas entradas y salidas como fueron pedidas.*

A partir de ahora veremos la arquitectura del programa, cabe aclarar que dicho programa fue realizado con una máquina de estado de la forma que vimos en clases, con sus respectivas "*Lógica de Estado*" y "*Lógica de salida*".

```
15 type estado is(inicio, activar_leer_pc, activar_esc_pc, cambiar_pc, reset_pc, espera, escribir_ram, activar_leer_ram, activar_esc_ram, leer_ram,
16                mostrar_salida, activar_leer_ri, activar_esc_ri, leer_ri, escribir_ri, activar_leer_op1, activar_esc_op1, escribir_op1, leer_op1, incrementar_pc,
17                activar_esc_data, activar_leer_data, escribir_data, leer_data, activar_leer_op2, activar_esc_op2, escribir_op2, leer_op2,
18                activar_carga_alu, activar_esc_resultado, activar_leer_resultado, escribir_resultado, leer_resultado,
19                activar_esc_status, activar_leer_status, escribir_status, leer_status,
20                activar_esc_salida, activar_leer_salida, escribir_salida, leer_salida, desactivar_salida);
```

**Estados utilizados.*

```
22 signal estadoActual: estado := inicio;
23 signal estadoSiguiente: estado;
24
25 signal en_ri, wr_ri, en_salida, wr_salida, en_pc, wr_pc, en_op1, wr_op1, en_op2, wr_op2, en_resultado, wr_resultado, en_status, wr_status, en_data, wr_data,
26        en_ram, wr_ram, op_a_cargar, carry_alu , ov_alu, zeta_alu, mostrar: std_logic:='0';
27
28 signal in_ram, out_ram, out_pc, in_ri, out_ri, in_op1, out_op1, in_op2, out_op2, in_data, out_data, in_resultado, out_resultado, in_status, out_status,
29        in_salida, out_salida: std_logic_vector (7 downto 0);
30
31 signal donde_leer: std_logic_vector (1 downto 0):= "00";
32 signal codigo_operacion_alu: std_logic_vector (2 downto 0):= "000";
33 signal pc_nuevo: std_logic_vector (5 downto 0):="000000";
34 signal in_pc, posicion_ram, posicion_actual: std_logic_vector (7 downto 0):="00000000";
```

**Señales utilizadas.*

```
37 alias clk: std_logic is control(0);
38 alias load: std_logic is control(1);
39 alias reset: std_logic is control(2);
```

**Alias utilizados para la separación de la señal de control.*

```
41  component alu is
42      port(
43          clk: in std_logic:='0';
44          carry,ov,zeta: out std_logic:='0';
45          op1: in std_logic_vector (7 downto 0):="00000000";
46          op2: in std_logic_vector(7 downto 0):="00000000";
47          cod_op: in std_logic_vector(2 downto 0):="000";
48          resultado: out std_logic_vector(7 downto 0):="00000000");
49  end component alu;
50  component registro8b is
51      port(
52          clk,en,wr: in std_logic:='0';
53          data_in: in std_logic_vector (7 downto 0):="00000000";
54          data_out: out std_logic_vector(7 downto 0):="00000000");
55  end component registro8b;
56  component memoria8b is
57      port(
58          clk,en,wr: in std_logic:='0';
59          data_in: in std_logic_vector (7 downto 0):="00000000";
60          data_out: out std_logic_vector(7 downto 0):="00000000";
61          direcciones: in std_logic_vector (7 downto 0));
62  end component memoria8b;
```

**Implementación de los programas (alu, registro8b y memoria8b) como componentes.*

A continuación comienza el funcionamiento del programa en sí:

```
66  modulo_alu: alu port map(clk=>clk,
67                           resultado=>in_resultado,
68                           op1=>out_op1, op2=>out_op2,
69                           cod_op=>codigo_operacion_alu,
70                           carry=>carry_alu, ov=>ov_alu, zeta=>zeta_alu);
71
72  r_operador1: registro8b port map (clk=>clk, en=>en_op1, wr=>wr_op1,
73                                    data_in=>in_op1,
74                                    data_out=>out_op1);
75
76  r_operador2: registro8b port map (clk=>clk, en=>en_op2, wr=>wr_op2,
77                                    data_in=>in_op2,
78                                    data_out=>out_op2);
79
80  r_resultado: registro8b port map (clk=>clk, en=>en_resultado, wr=>wr_resultado,
81                                    data_in=>in_resultado,
82                                    data_out=>out_resultado);
83
84  r_status: registro8b port map (clk=>clk, en=>en_status, wr=>wr_status,
85                                 data_in=>in_status,
86                                 data_out=>out_status);
87
88  r_ri: registro8b port map (clk=>clk, en=>en_ri, wr=>wr_ri,
89                             data_in=>in_ri,
90                             data_out=>out_ri);
91
92  r_data: registro8b port map (clk=>clk, en=>en_data, wr=>wr_data,
93                               data_in=>in_data,
94                               data_out=>out_data);
95
96  r_pc: registro8b port map (clk=>clk, en=>en_pc, wr=>wr_pc,
97                             data_in=>in_pc,
98                             data_out=>out_pc);
99
100 r_salida: registro8b port map (clk=>clk, en=>en_salida, wr=>wr_salida,
101                                data_in=>in_salida,
102                                data_out=>out_salida);
103
104 r_ram: memoria8b port map (clk=>clk, direcciones=>posicion_ram, en=>en_ram, wr=>wr_ram,
105                            data_in=>in_ram,
106                            data_out=>out_ram);
```

**Unión entre entradas y salidas de los componentes con las respectivas señales de nuestro programa general.*

```
logicaEstado: process (clk)
begin
   if clk'event and clk='1' then
      if reset='0' then
         estadoSiguiente<=reset_pc;
      else
         case estadoActual is
            when reset_pc=>estadoSiguiente<=inicio;
            when inicio=>estadoSiguiente<=espera;
            when espera=>estadoSiguiente<=activar_leer_pc;
            when activar_leer_pc=>
               if load='1' then
                  estadoSiguiente<=escribir_ram;
               elsif load='0' then
                  estadoSiguiente<=activar_leer_ram;
               else
                  estadoSiguiente<=reset_pc;
               end if;
            when activar_leer_ram=>estadoSiguiente<=leer_ram;
            when escribir_ram=>estadoSiguiente<=incrementar_pc;
            when leer_ram=>estadoSiguiente<=activar_esc_ri;
            when activar_esc_ri=>estadoSiguiente<=activar_leer_ri;
            when activar_leer_ri=>estadoSiguiente<=leer_ri;
            when leer_ri=>estadoSiguiente<=incrementar_pc;
```

**Comienzo de la lógica de estado, con las instrucciones básicas para el funcionamiento general del programa, como la utilización del control (clk, load, reset) además de la utilización básica de los componentes "memoria8b" y "registro8b).*

A continuación, veremos las instrucciones pedidas, luego de eso, como fueron implementadas en la lógica de estado para su utilización.

Código Instrucción								Descripción
0	0	0	0	1	0	0	0	Cargar operador1 (leer el operador X que se encuentre en memoria, cargarlo en Data y luego pasarlo a Op1)
X	X	X	X	X	X	X	X	
0	0	0	1	0	0	0	0	Cargar operador2 (leer el operador Y que se encuentre en memoria, cargarlo en Data y luego pasarlo a Op1)
Y	Y	Y	Y	Y	Y	Y	Y	
0	0	1	0	0	-	-	-	Sumar operador1 y operador2
0	0	1	1	0	-	-	-	Restar operador1 y operador2
0	0	1	0	1	-	-	-	AND operador1 y operador2
0	0	1	1	1	-	-	-	OR operador1 y operador2
0	1	0	0	0	-	-	-	SHL desplazamiento lógico a la izquierda de operador1
0	1	0	0	1	-	-	-	SHR desplazamiento lógico a la derecha de operador1
0	1	0	1	0	-	-	-	NOT negación de operador 1
0	1	0	1	1	-	-	-	leer el registro resultado, cargarlo en Data y luego pasarlo a Op1
0	1	1	0	0	-	-	-	Sacar resultado por la salida
0	1	1	0	1	-	-	-	Sacar status por la salida
0	1	1	1	0	-	-	-	leer el Data_In, cargarlo en Data y luego pasarlo a Op1
0	1	1	1	1	-	-	-	leer el Data_In, cargarlo en Data y luego pasarlo a Op2
1	0	A	A	A	A	A	A	Colocar PC en valor AAAAAA

```
133     when incrementar_pc=>
134        case out_ri is
135           when "00001000"=>
136              estadoSiguiente<=activar_esc_data;
137              op_a_cargar<='0';
138              donde leer<="00";
```

* *Instrucción 1.*

```
139           when "00010000"=>
140              estadoSiguiente<=activar_esc_data;
141              op_a_cargar<='1';
142              donde leer<="00";
```

* *Instrucción 2.*

```
143           when "00100000"=>
144              estadoSiguiente<=activar_carga_alu;
145              codigo operacion alu<="000";
```

* *Instrucción 3. (podemos observar la utilización del componente "alu").*

```
146           when "00110000"=>
147              estadoSiguiente<=activar_carga_alu;
148              codigo operacion alu<="001";
```

* *Instrucción 4.*

```
149           when "00101000"=>
150              estadoSiguiente<=activar_carga_alu;
151              codigo_operacion_alu<="011";
```

* *Instrucción 5.*

```
152           when "00111000"=>
153              estadoSiguiente<=activar_carga_alu;
154              codigo_operacion_alu<="010";
```

**Instrucción 6.*

```
155           when "01000000"=>
156              estadoSiguiente<=activar_carga_alu;
157              codigo_operacion_alu<="100";
```

* *Instrucción 7.*

```
158           when "01001000"=>
159              estadoSiguiente<=activar_carga_alu;
160              codigo_operacion_alu<="101";
```

**Instrucción 8.*

```
161           when "01010000"=>
162              estadoSiguiente<=activar_carga_alu;
163              codigo operacion alu<="110";
```

**Instrucción 9.*

```
164           when "01011000"=>
165              estadoSiguiente<=activar_esc_data;
166              donde_leer<="01";
167              op_a_cargar<='0';
```

**Instrucción 10.*

```
168           when "01100000"=>
169              estadoSiguiente<=activar_esc_salida;
170              mostrar<='0';
```

**Instrucción 11.*

```
171           when "01101000"=>
172              estadoSiguiente<=activar_esc_salida;
173              mostrar<='1';
```

**Instrucción 12.*

```
175             when "01110000"=>
176                estadoSiguiente<=activar_esc_data;
177                donde_leer<="10";
178                op_a_cargar<='0';
```

Instrucción 13.

```
179             when "01111000"=>
180                estadoSiguiente<=activar_esc_data;
181                donde_leer<="10";
182                op_a_cargar<='1';
```

Instrucción 14.

```
184             when others=>
185                if out_ri>="10000000" then
186                   if out_ri>="11000000" then
187                      estadoSiguiente<=espera;
188                   else
189                      estadosiguiente<=activar_esc_pc;
190                      pc_nuevo<=out_ri (5 downto 0);
191                   end if;
192                else
193                   estadoSiguiente<=espera;
194                end if;
195          end case;
```

Instrucción 15.

```
197             when activar_esc_pc=>estadoSiguiente<=cambiar_pc;
198             when cambiar_pc=>estadoSiguiente<=activar_leer_pc;
199             when activar_esc_data=>estadoSiguiente<=escribir_data;
200             when escribir_data=>estadoSiguiente<=activar_leer_data;
201             when activar_leer_data=>
202                if op_a_cargar='0' then
203                   estadoSiguiente<=activar_esc_op1;
204                else
205                   estadoSiguiente<=activar_esc_op2;
206                end if;
207             when activar_esc_op1=>estadoSiguiente<=escribir_op1;
208             when activar_esc_op2=>estadoSiguiente<=escribir_op2;
209
210             when escribir_op1=>estadoSiguiente<=activar_leer_op1;
211             when escribir_op2=>estadoSiguiente<=activar_leer_op2;
212
213             when activar_leer_op1=>estadoSiguiente<=leer_op1;
214             when activar_leer_op2=>estadoSiguiente<=leer_op2;
215
216             when leer_op1=>estadoSiguiente<=activar_leer_pc;--mostrar_salida;
217             when leer_op2=>estadoSiguiente<=activar_leer_pc;--mostrar_salida;
218
219             when activar_carga_alu=>estadoSiguiente<=activar_esc_resultado;
220             when activar_esc_resultado=>estadoSiguiente<=escribir_resultado;
221             when escribir_resultado=>estadoSiguiente<=activar_esc_status;
222             when activar_esc_status=>estadoSiguiente<=escribir_status;
223             when escribir_status=>estadoSiguiente<=activar_leer_resultado;
224             when activar_leer_resultado=>estadoSiguiente<=activar_leer_status;
225             when activar_leer_status=>estadoSiguiente<=activar_leer_pc;--mostrar_salida;
226
227             when mostrar_salida=>estadoSiguiente<=   activar_leer_pc;
228                                                   --incrementar_pc;
229             when activar_esc_salida=>estadoSiguiente<=escribir_salida;
230             when escribir_salida=>estadoSiguiente<=activar_leer_salida;
231             when activar_leer_salida=>estadoSiguiente<=leer_salida;
232             when leer_salida=>estadoSiguiente<=desactivar_salida;
233             when desactivar_salida=>estadoSiguiente<=activar_leer_pc;
234
235             when others=>estadoSiguiente<=inicio;
236          end case;
237       end if;
238    end if;
239
240 end process logicaEstado;
```

Aquí observamos los estados restantes de la lógica de estados, donde utilizamos las señales para escribir o leer los operadores, enviamos los resultados a la salida, etc.

A partir de este punto observaremos todo el funcionamiento de la lógica de salida.

```
247  logicaSalida: process (estadoActual)
248    variable en_pc_v: std_logic:=en_pc;
249    variable wr_pc_v: std_logic:=wr_pc;
250    variable en_ram_v: std_logic:=en_ram;
251    variable wr_ram_v: std_logic:=wr_ram;
252    variable en_ri_v: std_logic:=en_ri;
253    variable wr_ri_v: std_logic:=wr_ri;
254    variable en_salida_v: std_logic:=en_salida;
255    variable wr_salida_v: std_logic:=wr_salida;
256    variable en_data_v: std_logic:=en_data;
257    variable wr_data_v: std_logic:=wr_data;
258    variable en_op1_v: std_logic:=en_op1;
259    variable wr_op1_v: std_logic:=wr_op1;
260    variable en_op2_v: std_logic:=en_op2;
261    variable wr_op2_v: std_logic:=wr_op2;
262    variable en_status_v: std_logic:=en_status;
263    variable wr_status_v: std_logic:=wr_status;
264    variable en_resultado_v: std_logic:=en_resultado;
265    variable wr_resultado_v: std_logic:=wr_resultado;
```

**Variables que vamos a utilizar.*

```
267  begin
268     case estadoActual is
269        when inicio=>
270        when espera=>
271           en_ri_v:='0';
272           wr_ri_v:='0';
273           en_salida_v:='0';
274           wr_salida_v:='0';
275           en_pc_v:='0';
276           wr_pc_v:='0';
277           en_ram_v:='0';
278           wr_ram_v:='0';
279
280           en_data_v:='0';
281           wr_data_v:='0';
282           en_resultado_v:='0';
283           wr_resultado_v:='0';
284           en_op1_v:='0';
285           wr_op1_v:='0';
286           en_op2_v:='0';
287           wr_op2_v:='0';
288           en_status_v:='0';
289           wr_status_v:='0';
```

**Configuración de salidas en el estado "estadoActual".*

```
291        when reset_pc=>
292           en_pc_v:='1';
293           wr_pc_v:='1';
294           in_pc<="00000000";
295           posicion_ram<="00000000";
296           posicion_actual<="00000000";
```

**Configuración de salidas en el estado "reset_pc".*

```
298         when activar_leer_pc=>
299            en_pc_v:='1';
300            wr_pc_v:='0';
```
*Configuración de salidas en el estado "activar_leer_pc".

```
302         when activar_esc_pc=>
303            en_pc_v:='1';
304            wr_pc_v:='1';
```
*Configuración de salidas en el estado "activar_escribir_pc".

```
306         when cambiar_pc=>
307            en_pc_v:='1';
308            wr_pc_v:='1';
309            in_pc<="00"&pc_nuevo;
```
*Configuración de salidas en el estado "cambiar_pc".

```
311         when escribir_ram=>
312            en_ram_v:='1';
313            wr_ram_v:='1';
314            in_ram<=data_in;
315
316            posicion_ram<=out_pc;
317            posicion_actual<=out_pc;
```
*Configuración de salidas en el estado "escribir_ram".

```
319         when activar_leer_ram=>
320            en_ram_v:='1';
321            wr_ram_v:='0';
```
*Configuración de salidas en el estado "activar_leer_ram".

```
323         when leer_ram=>
324            in_ri<=out_ram;
325
326            posicion_ram<=out_pc;
327            posicion_actual<=out_pc;
```
*Configuración de salidas en el estado "leer_ram".

```
329         when activar_leer_ri=>
330            en_ri_v:='1';
331            wr_ri_v:='0';
```
*Configuración de salidas en el estado "activar_leer_ri".

```
333         when activar_esc_ri=>
334            en_ri_v:='1';
335            wr_ri_v:='1';
```
*Configuración de salidas en el estado "activar_esc_ri".

```
337         when leer_ri=>
338            --salida<=out_ri;
339            en_ri_v:='1';
340            wr_ri_v:='0';
```
*Configuración de salidas en el estado "leer_ri".

```
342         when activar_esc_op1=>
343            en_op1_v:='1';
344            wr_op1_v:='1';
```
*Configuración de salidas en el estado "activar_esc_op1".

```
346         when activar_leer_op1=>
347            en_op1_v:='1';
348            wr_op1_v:='0';
```
*Configuración de salidas en el estado "activar_leer_op1".

```
350         when activar_esc_op2=>
351            en_op2_v:='1';
352            wr_op2_v:='1';
```
*Configuración de salidas en el estado "activar_esc_op2".

```
354         when activar_leer_op2=>
355            en_op2_v:='1';
356            wr_op2_v:='0';
```
*Configuración de salidas en el estado "activar_leer_op2".

```
358         when activar_esc_data=>
359            en_data_v:='1';
360            wr_data_v:='1';
361
362            en_ram_v:='1';
363            wr_ram_v:='0';
364
365            en_resultado_v:='1';
366            wr_resultado_v:='0';
```
*Configuración de salidas en el estado "activar_esc_data".

```
368         when activar_leer_data=>
369            en_data_v:='1';
370            wr_data_v:='0';
```
*Configuración de salidas en el estado "activar_leer_data".

```
372         when leer_data=>
373            en_data_v:='1';
374            wr_data_v:='0';
```
*Configuración de salidas en el estado "leer_data".

```
376         when escribir_data=>
377
378            case donde_leer is
379               when "00"=> in_data<=out_ram;
380               when "01"=> in_data<=out_resultado;
381               when "10"=> in_data<=data_in;
382               when others=>
383            end case;
384            --in data<=out ram;
```
*Configuración de salidas en el estado "escribir_data".

```
386         when escribir_op1=>
387            in_op1<=out_data;
388
389            en_ri_v:='1';
390            wr_ri_v:='1';
391            in_ri<="00000000";
```

Configuración de salidas en el estado "escribir_op1".

```
393         when leer_op1=>
394            en_op1_v:='1';
395            wr_op1_v:='0';
```

Configuración de salidas en el estado "leer_op1".

```
397         when leer_op2=>
398            en_op2_v:='1';
399            wr_op2_v:='0';
```

Configuración de salidas en el estado "leer_op2".

```
401         when escribir_op2=>
402            in_op2<=out_data;
403
404            en_ri_v:='1';
405            wr_ri_v:='1';
406            in ri<="00000000";
```

Configuración de salidas en el estado "escribir_op2".

```
408         when activar_carga_alu=>
409            en_op1_v:='1';
410            wr_op1_v:='0';
411            en_op2_v:='1';
412            wr_op2_v:='0';
```

Configuración de salidas en el estado "activar_carga_alu".

```
414         when activar_esc_resultado=>
415            en_resultado_v:='1';
416            wr_resultado_v:='1';
```

Configuración de salidas en el estado "activar_esc_resultado".

```
418         when activar_esc_status=>
419            en_status_v:='1';
420            wr_status_v:='1';
```

Configuración de salidas en el estado "activar_esc_status".

```
422         when activar_leer_resultado=>
423            en_resultado_v:='1';
424            wr_resultado_v:='0';
```

Configuración de salidas en el estado "activar_leer_resultado".

```
426         when activar_leer_status=>
427            en_status_v:='1';
428            wr_status_v:='0';
```

Configuración de salidas en el estado "activar_leer_status".

```
430         when escribir_resultado=>
431            en_resultado_v:='1';
432            wr_resultado_v:='1';
```
*Configuración de salidas en el estado "escribir_resultado".

```
434         when escribir_status=>
435            en_status_v:='1';
436            wr_status_v:='1';
437            in_status<="00000"&carry_alu&ov_alu&zeta_alu;
```
*Configuración de salidas en el estado "escribir_status".

```
439         when leer_resultado=>
440            en_resultado_v:='1';
441            wr_resultado_v:='0';
```
*Configuración de salidas en el estado "leer_resultado".

```
443         when leer_status=>
444            en_status_v:='1';
445            wr_status_v:='0';
```
*Configuración de salidas en el estado "leer_status".

```
447         when activar_esc_salida=>
448            en_salida_v:='1';
449            wr_salida_v:='1';
450
451            en_ri_v:='1';
452            wr_ri_v:='0';
```
*Configuración de salidas en el estado "activar_esc_salida".

```
454         when escribir_salida=>
455            if mostrar='0' then
456               in_salida<=out_resultado;
457            else
458               in_salida<=out_status;
459            end if;
460            en_salida_v:='1';
461            wr_salida_v:='1';
```
*Configuración de salidas en el estado "escribir_salida".

```
463         when activar_leer_salida=>
464            en_salida_v:='1';
465            wr_salida_v:='0';
```
*Configuración de salidas en el estado "activar_leer_salida".

```
467         when leer_salida=>
468            en_salida_v:='1';
469            wr_salida_v:='0';
470            salida<=out_salida;
```
*Configuración de salidas en el estado "leer_salida".

```
472         when desactivar_salida=>
473            en_salida_v:='0';
474            wr_salida_v:='0';
475            salida<=out_salida;
```

**Configuración de salidas en el estado "desactivar_salida".*

```
478         when incrementar_pc=>
479            en_pc_v:='1';
480            wr_pc_v:='1';
481
482            in_pc<=posicion_actual+1;
483            if posicion_actual="00111111" then
484               in_pc<="00000000";
485            end if;
486
487            en_ram_v:='0';
488            wr_ram_v:='0';
489            in_ram<="ZZZZZZZZ";
490
491         when others=>
```

**Configuración de salidas en el estado "incrementar_pc".*

```
512      end case;
513
514      en_pc<=en_pc_v;
515      wr_pc<=wr_pc_v;
516      en_ram<=en_ram_v;
517      wr_ram<=wr_ram_v;
518      en_ri<=en_ri_v;
519      wr_ri<=wr_ri_v;
520      en_salida<=en_salida_v;
521      wr_salida<=wr_salida_v;
522      en_data<=en_data_v;
523      wr_data<=wr_data_v;
524      en_op1<=en_op1_v;
525      wr_op1<=wr_op1_v;
526      en_op2<=en_op2_v;
527      wr_op2<=wr_op2_v;
528      en_status<=en_status_v;
529      wr_status<=wr_status_v;
530      en_resultado<=en_resultado_v;
531      wr_resultado<=wr_resultado_v;
532   end process logicaSalida;
533
534   end architecture cpu;
```

**En esta parte indicamos que variable corresponde a determinada señal, luego de eso terminamos con la lógica de salida. Aquí termina toda la configuración del programa.*

Configuración de la simulación TestBench:

```
54   init : PROCESS
55   -- variable declarations
56   BEGIN
57      control(2)<='0';
58      wait for 980ns;
59      control(2)<='1';
60      wait for 1430ns;
61      control(2)<='0';
62      wait for 50ns;
63      control(2)<='1';
64      --wait for 4000ns;
65   WAIT;
66   END PROCESS init;
```

**Proceso "init", donde controlamos el reset.*

```
67   clock : PROCESS
68   -- optional sensitivity list
69   -- (        )
70   -- variable declarations
71   BEGIN
72      control(0)<='0';
73      wait for 10ns;
74      control(0)<='1';
75      wait for 10ns;
76   END PROCESS clock;
```

**Proceso "clock" donde vamos activando el clk.*

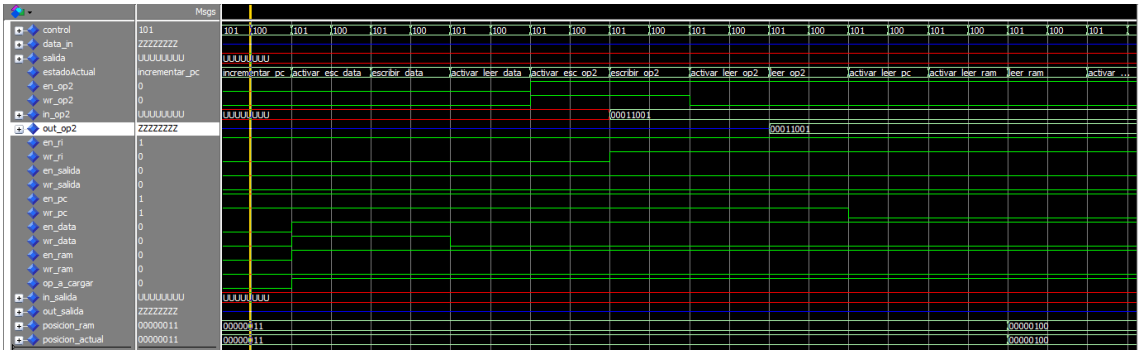
Las siguientes imágenes corresponden al proceso del programa en general, donde le pedimos simular todas las instrucciones que fueron pedidas.

```
77   programa: PROCESS
78   BEGIN
79      data_in<="ZZZZZZZZ";
80      wait for 900ns;
81      control(1)<='1';
82      wait for 150ns;
83      data_in<="00001000"; -- operacion cargar op1
84      wait for 20ns;
85      data_in<="ZZZZZZZZ";
86      wait for 60ns;
87      data_in<="00101100"; -- operador a cargar en op1
88      wait for 20ns;
89      data_in<="ZZZZZZZZ";
90      wait for 60ns;
91      data_in<="00010000"; -- operacion cargar op2
92      wait for 20ns;
93      data_in<="ZZZZZZZZ";
94      wait for 60ns;
95      data_in<="00011001"; -- operador a cargar en op2
96      wait for 20ns;
97      data_in<="ZZZZZZZZ";
98      wait for 60ns;
99      data_in<="00100000"; -- sumar
100     wait for 20ns;
```

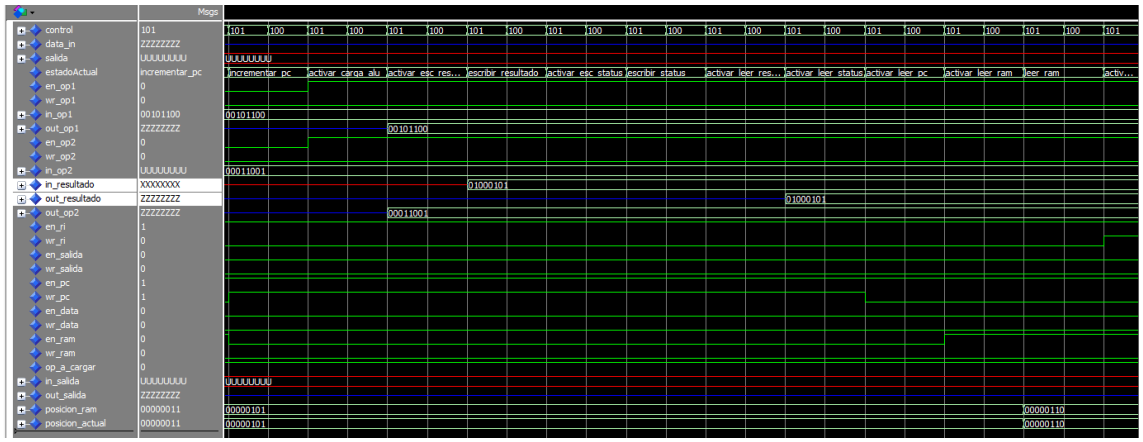
```
101     data_in<="ZZZZZZZZ";
102     wait for 60ns;
103     data_in<="00110000"; -- restar
104     wait for 20ns;
105     data_in<="ZZZZZZZZ";
106     wait for 60ns;
107     data_in<="00101000"; -- and
108     wait for 20ns;
109     data_in<="ZZZZZZZZ";
110     wait for 60ns;
111     data_in<="00111000"; -- or
112     wait for 20ns;
113     data_in<="ZZZZZZZZ";
114     wait for 60ns;
115     data_in<="01000000"; -- shl
116     wait for 20ns;
117     data_in<="ZZZZZZZZ";
118     wait for 60ns;
119     data_in<="01001000"; -- shr
120     wait for 20ns;
121     data_in<="ZZZZZZZZ";
122     wait for 60ns;
123     data_in<="01010000"; -- not
124     wait for 20ns;
125     data_in<="ZZZZZZZZ";
126     wait for 60ns;
127     data_in<="01011000"; -- resultado a op1
128     wait for 20ns;
129     data_in<="ZZZZZZZZ";
130     wait for 60ns;
131     data_in<="01100000"; -- mostrar resultado
132     wait for 20ns;
133     data_in<="ZZZZZZZZ";
134     wait for 60ns;
135     data_in<="01101000"; -- mostrar status
136     wait for 20ns;
137     data_in<="ZZZZZZZZ";
138     wait for 60ns;
139     data_in<="01110000"; -- cargar op1 con data_in
140     wait for 20ns;
141     data_in<="ZZZZZZZZ";
142     wait for 60ns;
143     data_in<="01111000"; -- cargar op2 con data_in
144     wait for 20ns;
145     data_in<="ZZZZZZZZ";
146     wait for 60ns;
147     data_in<="10000001"; -- cambiar pc a 000001
148     wait for 20ns;
149     data_in<="ZZZZZZZZ";
150     wait for 60ns;
151     control(1)<='0';
152     wait for 4120ns;
153     data_in<="11111110";
154     wait for 80ns;
155     data_in<="ZZZZZZZZ";
156     wait for 130ns;
157     data_in<="00000001";
158     wait for 100ns;
159     data_in<="ZZZZZZZZ";
160     wait;
161
162 wait;
163 END PROCESS programa;
164 END cpu_nuevo_arch;
```

Resultados de la simulación:

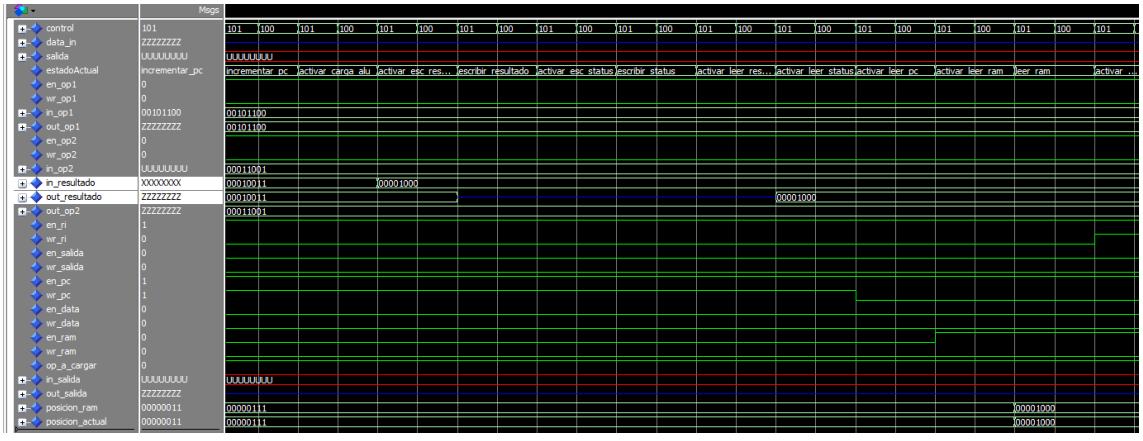
Instrucción 1.



Instrucción 2.

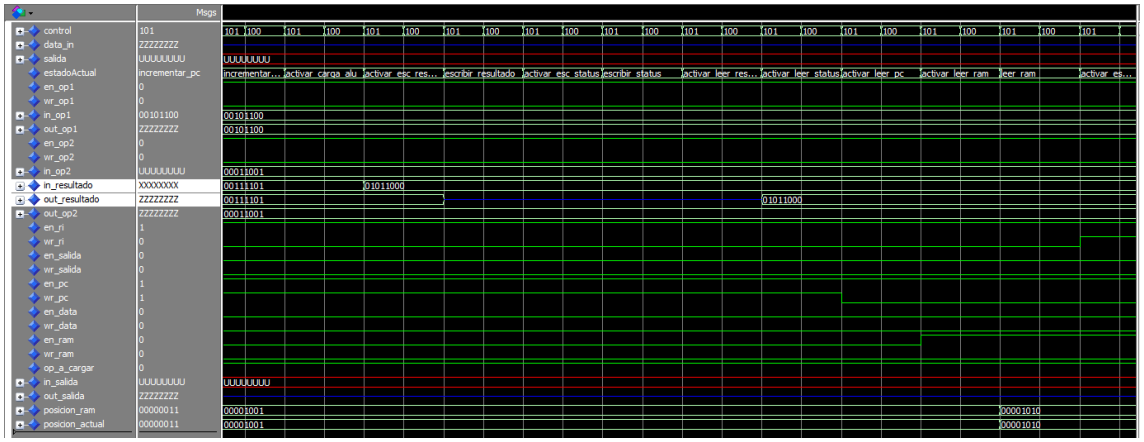


Instrucción 3.



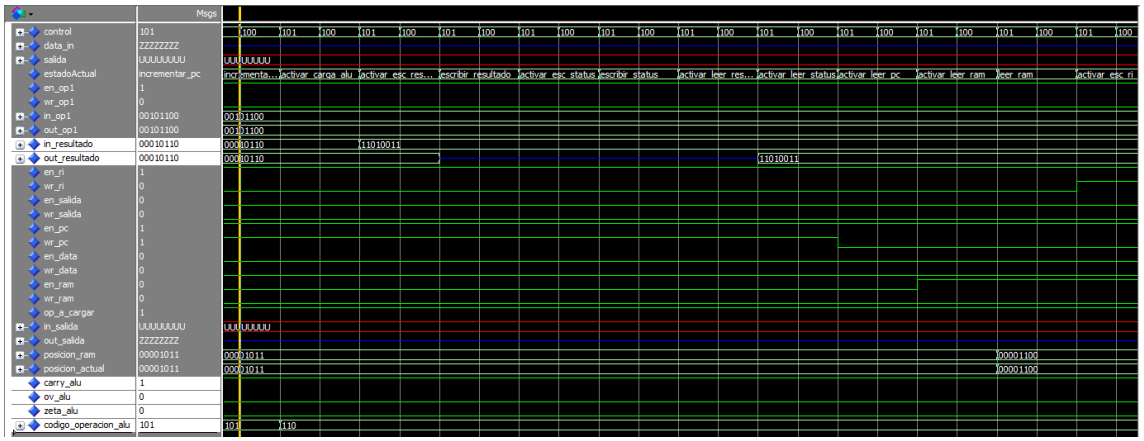
Instrucción 4.

Instrucción 5.

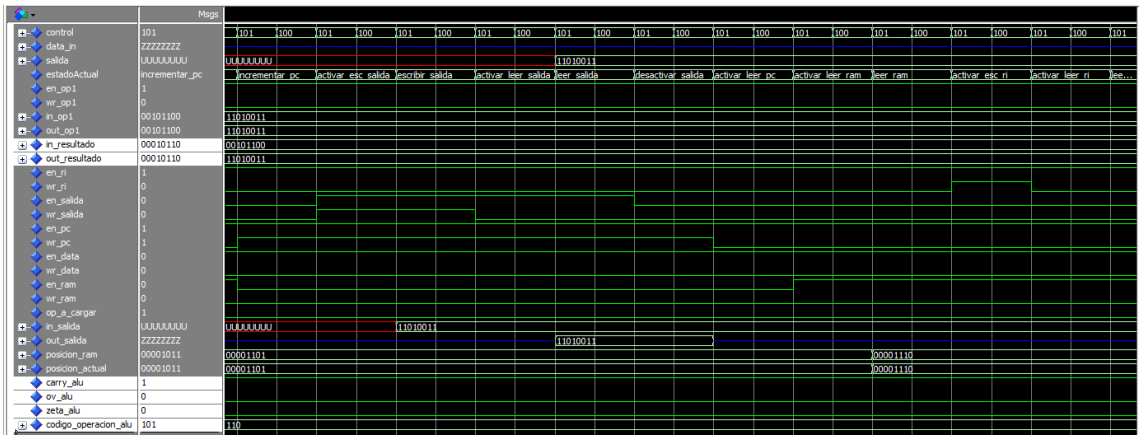


Instrucción 6.

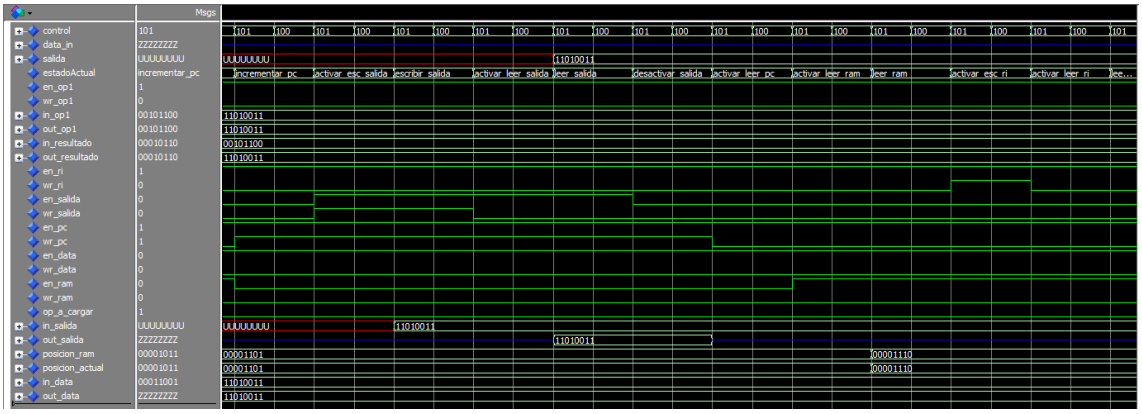
Instrucción 7.



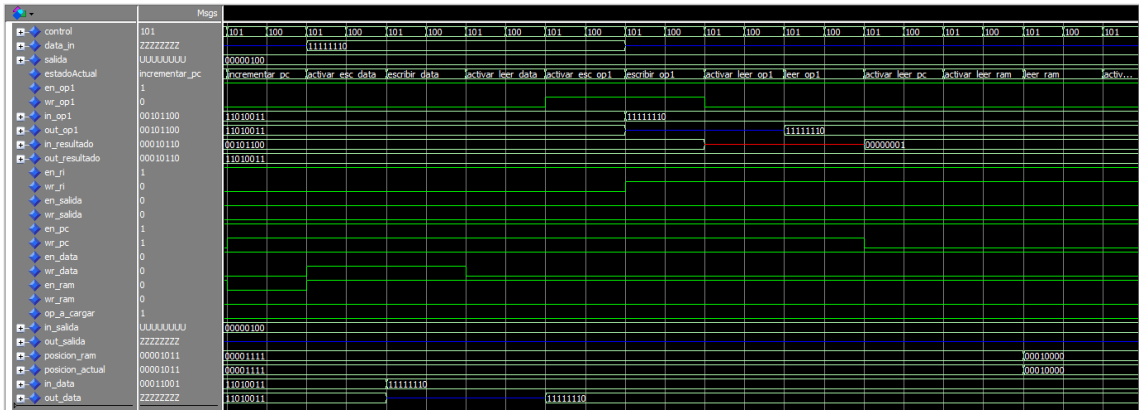
Instrucción 8.



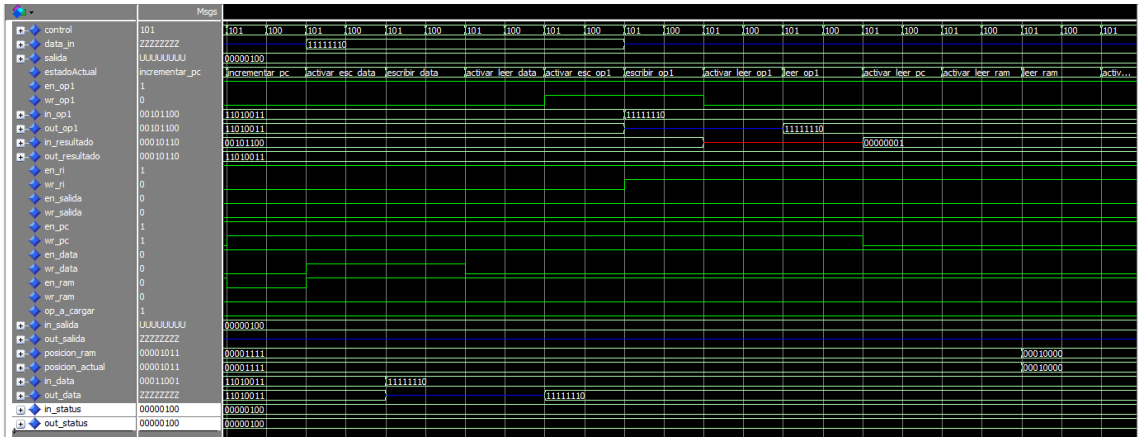
**Instrucción 9.*



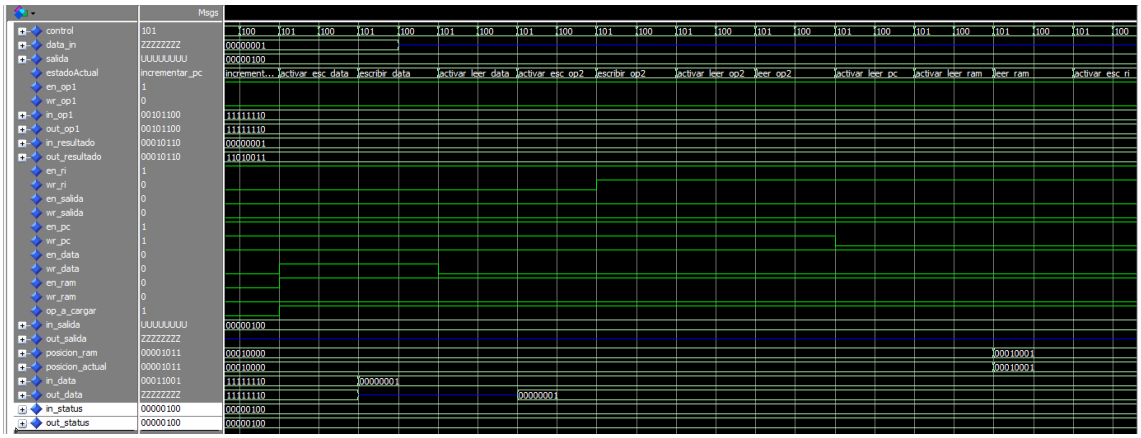
**Instrucción 10.*



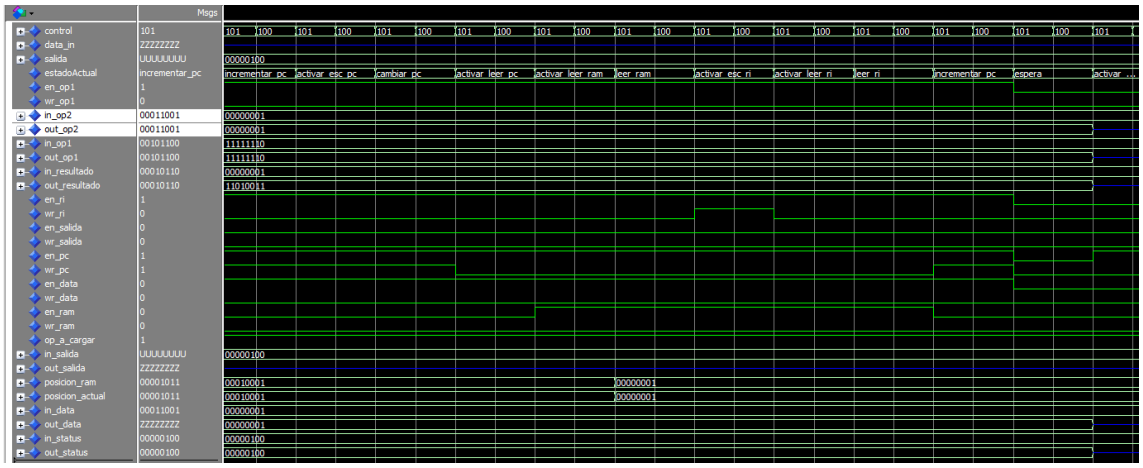
**Instrucción 11.*



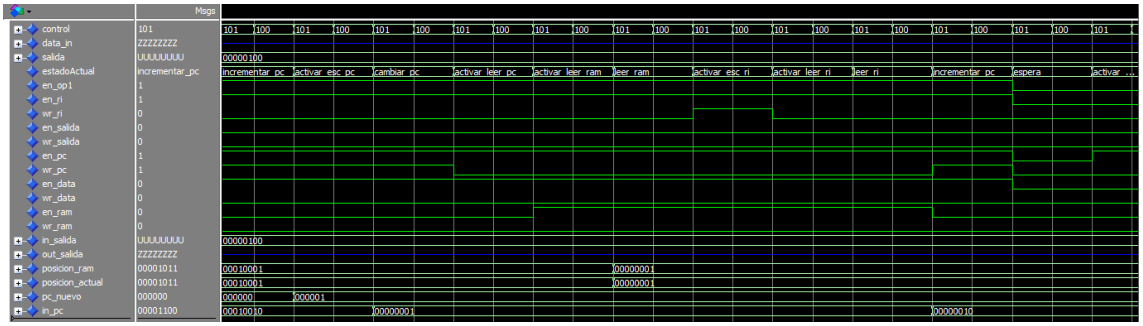
**Instrucción 12.*



Instrucción 13.



Instrucción 14.



Instrucción 15.