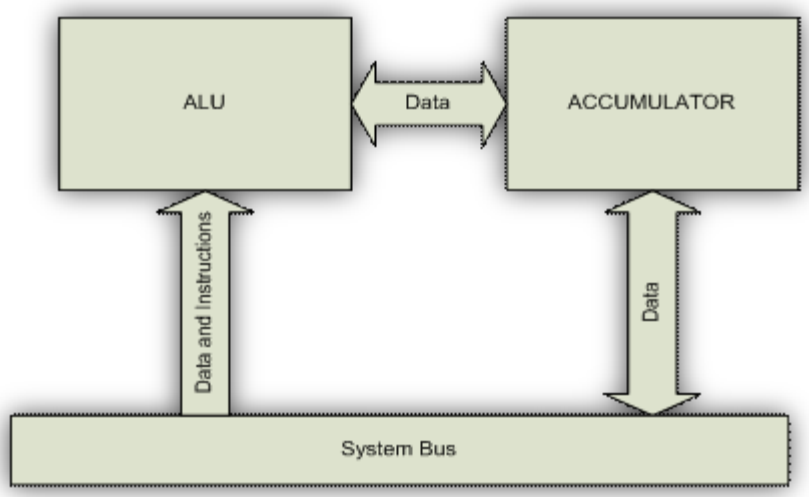
Diseño de un Microprocesador en VHDL



Miriam Puente García 71449758M
Jose Alberto Benítez Andrades 71454586A

Arquitectura e Ingeniería de Computadores

Ingeniería en Informática

Universidad de León

ÍNDICE

1.Diseño de un Microprocesador en VHDL

Gracias a nuestro conocimiento del lenguaje ***VHDL***, el cual hemos aprendido gracias a la realización de diversas prácticas para la asignatura de *Arquitectura e Ingeniería de Computadores*, y con ayuda de la búsqueda de información vía *Internet* y *Libros*, hemos podido obtener un diseño de un tipo de microprocesador.

La finalidad de este trabajo, era la de encontrar una aproximación a la arquitectura de computadores en sí, es decir, poder configurar los procesadores que actúan siguiendo un software. Para ello lo primero que hemos hecho ha sido buscar un tipo de microprocesador adecuado para la tarea que queremos realizar (diseño y funcionalidad del mismo). De todos los tipos de procesadores que hay, hemos elegido un microprocesador que posee un repertorio de instrucciones reducido y con un solo modo de direccionamiento, en este caso, directo (así evitamos la necesidad de introducir conceptos de "segundo orden", como el direccionamiento indexado o similares).

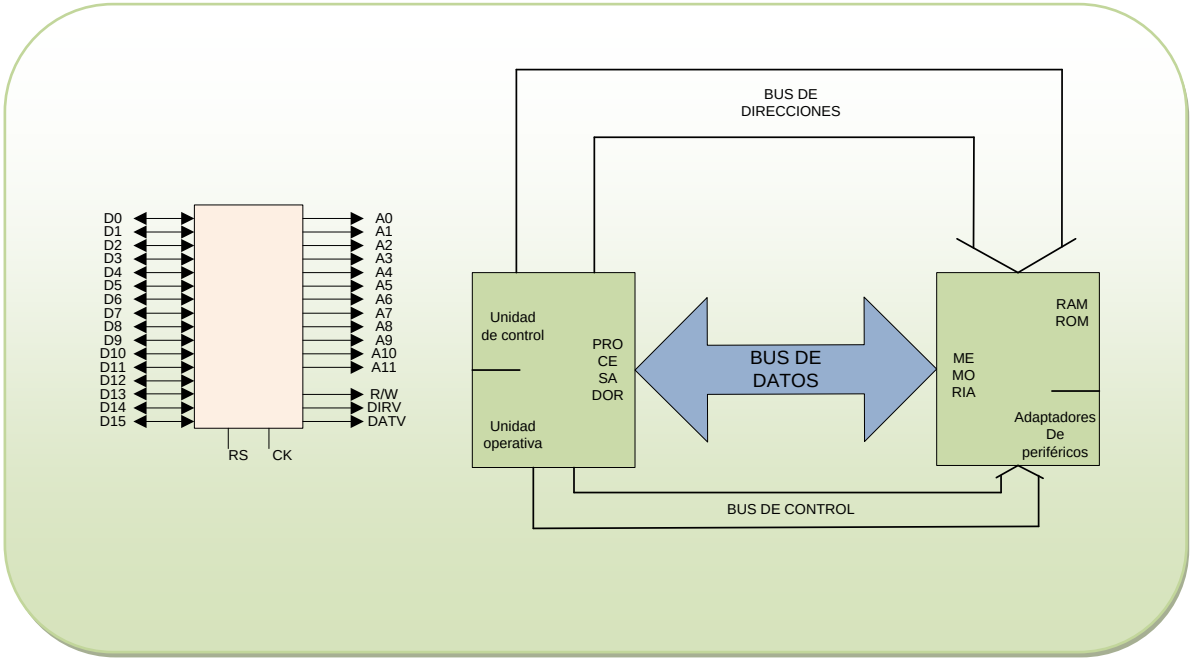
Nuestro microprocesador posee 16 instrucciones que se dividen en 4 grupos homogéneos:
- aritméticas: suma **ADD**, resta **SUB**, incrementar **INC** y decrementar **DEC**;
- lógicas: operaciones "**y**" **AND** y "**o**" **ORA**, inversión **INV** y desplazamiento **SRR**;
- transferencias: llevar dato a acumulador A **LDA** o a acumulador B **LDB**, borrado de dato **CLR** y almacenamiento del contador de programa **SPC**;
- y saltos: uno de ellos incondicional **JMP** y otros tres condicionados a los indicadores (resultado nulo **BRZ**, acarreo **BSC** y desbordamiento **BSV**).

2. Configuración del microprocesador

El microprocesador que hemos elegido para diseñar con VHDL, tendrá como característica el uso de palabras de 16 bits (datos e instrucciones). Así el intervalo de una palabra será de 0 a 65.535 y si es un número entero, entre -32.768 y 32.768. El código de instrucción son 4 de esos 16 bits, y los 12 restantes son para el direccionamiento del dato.

Los terminales exteriores del procesador serán:

- Bus de datos de 16 líneas: datos e instrucciones de 16 bits
- Bus de direcciones de 14 líneas: mapa de memoria de 4K
- Bus de control con las tres líneas básicas: R/W, DIRV y DATV
- Entradas de reloj CK, inicialización *(Reset)* RS y alimentación.

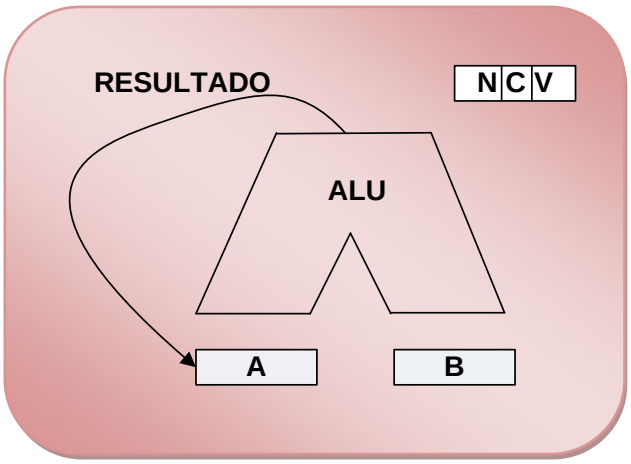


Un procesador se divide básicamente en 2 partes, una de ellas es la parte *operativa* y la otra es la de *control*.

Parte Operativa

Esta parte es la que realiza las operaciones y almacena los operandos y resultados. En el caso que nosotros presentamos tendremos 2 registros acumuladores que llamaremos **A** y **B**, una **ALU (Arithmetic Logic Unit)** capaz de sumar, restar y realizar operaciónes lógicas, y tres biestables que indican el resultado nulo **N**, el acarreo **C** y el desbordamiento **V**.

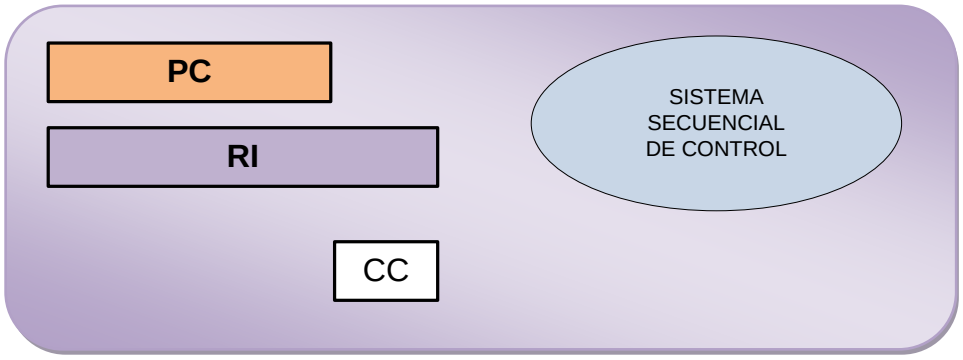
Los registros **A** y **B** junto con el acarreo **C** son los encargados de almacenar los resultados de las operaciones. También el indicador de acarreo **C** y de resultado nulo **N** se utilizan para hacer comparaciones de menor, igual, por medio de la instrucción resta **SUB** en las salidas. Esto ayuda a realizar operaciones de salto.



Unidad de Control

Desde esta unidad con la ayuda de un contador de programa **PC** que señala la dirección de la siguiente instrucción que se debe ejecutar, un registro de instrucciones **RI** que almacena la instrucción en ejecución y su circuito secuencial decodificador y controlador de la ejecución de instrucciones, realiza el control (**CC**).

Para no complicarnos, hemos hecho que cada instrucción utilice dos ciclos de reloj, en el primero realiza una búsqueda y en el segundo la operación que prescribe se ejecuta. Para diferenciar los 2 ciclos, se utiliza un biestable de tipo T.



3.Repertorio de instrucciones

Las instrucciones distinguen 2 partes, una es su código de instrucción de 4 bits **COD** y otra la dirección de memoria **DIR** de 12 bits, como bien expliqué anteriormente.

Hay varios tipos de instrucciones:

Instrucciones aritméticas y lógicas

De 2 operandos

- Las instrucciones que se aplican sobre dos operandos utilizan los 2 registros **A** y **B** y el resultado se almacena en **A**, además de en la dirección que indique la instrucción.

	CÓDIGO	OPERACIÓN	INDICADORES A LOS QUE AFECTAN
ADD	0000	A + B	Z (resultado nulo), C (acarreo) y V (desbordamiento)
SUB	0001	A – B	Z (resultado nulo), C (acarreo) y V (desbordamiento)
AND	0010	A “y” B	Z (resultado nulo)
ORA	0011	A “o” B	Z (resultado nulo)

respecto a las 4 instrucciones: resultado → registro **A**

resultado → memoria(**DIR**).

Primero se guardarán en los registros **A** y **B** los operandos, mediantes instrucciones de carga **LDA** y **LDB.** Durante el segundo ciclo de reloj de cada una de estas instrucciones, la **ALU** efectuará la operación que corresponda y, al finalizar dicho ciclo, el resultado presente en la salida de la **ALU** será almacenado, a la vez, en el registro **A** y en la memoria (en la dirección **DIR** contenida en la propia instrucción).

De un solo operando

Las instrucciones que se aplican sobre un solo operando utilizan el acumulador **A**, tanto como operando como para recoger el resultado, que, también, se almacena en la dirección de memoria **DIR** indicada en la instrucción.

	CÓDIGO	OPERACIÓN	INDICADORES A LOS QUE AFECTAN
INC	0100	A + 1	Z (resultado nulo)
DEC	0101	A – 1	Z (resultado nulo)
INV	0110	$\overline{A}$	Z (resultado nulo)
SRR	0111	A →	Z (resultado nulo), C (acarreo)

respecto a las 4 instrucciones: resultado → acumulador **A**

resultado → memoria(**DIR**).

Al igual que en el caso anterior, para las ocasiones en que no interese almacenar el resultado en memoria, se reservará una dirección de memoria (el «registro papelera»). El operando deberá encontrarse en el registro **A**, mediante una instrucción de carga (**LDA**) o como resultado de operaciones precedentes. Durante el segundo ciclo de reloj, la **ALU** efectuará la operación y, al finalizar el ciclo, el resultado será almacenado en el acumulador **A** y en la memoria (dirección **DIR**).

Descripción VHDL de la ALU (para las operaciones aritméticas y lógicas)

```
1   library ieee; use ieee.std_logic_1164.all; use ieee.std_logic_unsigned.all;
2     entity ALU is
3       port ( A, B : in std_logic_vector (15 downto 0); -- registros
4              COD  : in std_logic_vector (3 downto 0); -- código de instrucción
5              CC   : in std_logic; -- ciclo de aplicación (biestable T que controla los ciclos)
6              R    : out std_logic_vector (15 downto 0); -- resultado
7              C    : in std_logic; -- entrada de acarreo
8              Nulo, Carry, V_over : out std_logic ); -- indicadores
9     end ALU;
10    architecture OPERACIONES of ALU is
11    -- necesidad de una señal de resultado de un bit más para detectar acarreos
12    signal RR : STD_logic_vector(16 downto 0);
13    begin
14    -- operaciones de la ALU
15      process(COD, A, B, C, CC)
16      begin
17        if CC = '0' or COD(3) = '1' then RR <= (others => '0');
18        -- solo toman valor en el ciclo de ejecución de instrucciones de tipo a)
19        else
20          RR(16) <= '0'; -- por defecto para operaciones que no utilizan acarreo
21          case COD is - con el código de instrucción en los primeros 4 bits, seleccionamos la operación
22            when "0000" => RR <= '0' & A + B + C; -- suma (con acarreo)
23            when "0001" => RR <= '0' & A - B - C; -- resta (con acarreo)
24            when "0010" => RR(15 downto 0) <= A and B; -- and
25            when "0011" => RR(15 downto 0) <= A or B; -- or
26            when "0100" => RR <= '0' & A + 1; -- incrementar
27            when "0101" => RR <= '0' & A - 1; -- decrementar
28            when "0110" => RR(15 downto 0)<= not B; -- invertir
29            -- deplazamiento hacia la derecha
30            when "0111" => RR(15 downto 0) <= B(0) & B(15 downto 1);
31            when others => RR <= (others => '0');
32          end case;
33        end if;
34      end process;
35      -- resultado
```

```
36      R <= RR(15 downto 0);
37
38      -- indicadores
39      process(COD,RR, A, B, CC)
40      begin
41      Nulo <= '0'; Carry <= '0'; V_over <= '0'; -- por defecto
42        if CC = '1' and COD(3) = '0' then
43        -- solo toman valor en el ciclo de ejecución de instrucciones de tipo a)
44        -- resultado nulo
45        if RR(15 downto 0) = "0000000000000000" then Nulo <= '1'; end if;
46        -- acarreo
47        if  COD = "0000" or COD = "0001" then Carry <= RR(16); end if; -- suma o resta
48        if COD = "1111" then Carry <= A(0); end if; -- desplazamiento
49        -- desbordamiento (over-flow)
50        if COD = "0000" then -- operación de suma
51        if (A(15) = '0' and B(15) = '0' and RR(15) = '1')
52          or (A(15) = '1' and B(15) = '1' and RR(15) = '0') then V_over<= '1'; end if;
53        end if;
54        if COD = "0001" then -- operación de resta
55        if (A(15) = '0' and B(15) = '1' and RR(15) = '1')
56          or (A(15) = '1' and B(15) = '0' and RR(15) = '0') then V_over<= '1'; end if;
57        end if;
58        end if;
59      end process;
60
61  end OPERACIONES;
```

Instrucciones de transferencia con memoria

Sirven para llevar datos de memoria a los acumuladores (**LDA**, **LDB**), para borrar (**CLR**) una posición de memoria (se borran, también, a la vez, el acumulador **B** y el indicador de acarreo **C**) o para almacenar en memoria el contenido del contador de programa (**SPC**).

	CÓDIGO	OPERACIÓN	INDICADORES
LDA	1000	memoria(**DIR**) → A	No afectan a ninguno
LDB	1001	memoria(**DIR**) → B	No afectan a ninguno
CLR	1010	0 →memoria(**DIR**); 0 → B; 0 → C	C (acarreo)
SPC	1011	PC →memoria(**DIR**)	No afectan a ninguno

uno para la búsqueda de la instrucción y otro para la transferencia del dato desde o hacia la memoria; durante el segundo ciclo, se selecciona, a través del bus de direcciones, la posición de memoria indicada en la instrucción (**DIR**) y, al finalizar el ciclo, se ejecuta la transferencia del dato.

La instrucción de borrado puede utilizarse para borrar una posición de memoria (**DIR**) o para borrar el acumulador **B** o para poner a **0** el indicador de acarreo **C**; esto último es necesario hacerlo previamente a una operación de suma o resta para evitar que se añadan a ella acarreos de operaciones anteriores.

Instrucciones de salto

Estas instrucciones producen un salto en la ejecución del programa (en la sucesión de instrucciones que se están ejecutando), modificando el contenido del contador de programa PC, de forma que no apunte a la siguiente posición de memoria, sino a otra; esa nueva posición de memoria es la indicada (**DIR**) en la propia instrucción.

En el caso de la primera instrucción (**JMP**) el salto en el programa se produce siempre; las tres siguientes condicionan el salto a que el valor del correspondiente indicador (**N**, **C**, **V**) sea **1**.

	CÓDIGO	**OPERACIÓN**	**INDICADORES a los que afectan**
JMP	1100	**DIR** → PC (incondicional)	Ninguno
BRZ	1101	**DIR** → PC si Z = **1**	Ninguno
BRC	1110	**DIR** → PC si C = **1**	Ninguno
BRV	1111	**DIR** → PC si V = **1**	Ninguno

En el segundo ciclo de reloj, la primera instrucción transfiere **DIR** al contador de programa **PC**; las otras tres instrucciones hacen lo mismo, pero comprueban antes que el indicador correspondiente se encuentra a **1** (si su valor es **0**, no hacen nada):

- Un salto **BRZ** se produce si el resultado de la operación anterior es nulo (**Z = 1**); si se encuentra después de una instrucción de resta, el salto se produce cuando los dos operandos son iguales.
- Un salto **BRC** (**C = 1**) posterior a una instrucción de resta, se efectúa si el operando **B** es mayor que el **A**.
- Los saltos **BRV** (**V = 1**) sirven para supervisar el posible desbordamiento en las operaciones de suma o resta de números en codificación en complemento a 2.

Descripción VHDL

La descripción VHDL de la ALU se encuentra en el apartado anterior (unida a la descripción de las instrucciones aritméticas y lógicas cuya funcionalidad es realizada circuitalmente por la ALU).

El siguiente fichero completa la descripción global del «microprocesador elemental», incluyendo la ALU como componente; previamente a cada bloque de la descripción, un comentario explica y justifica su diseño.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4     entity MICRO is
5     port ( RS, CK : in std_logic;
6           Databus : inout std_logic_vector(15 downto 0);
7            Dirbus : out std_logic_vector(11 downto 0);
8   R_W, DIRV, DATV : out std_logic );
9     end MICRO;
10    architecture PROCESADOR of MICRO is
11    -- acumuladores, resultado e indicadores
12    signal A, B : std_logic_vector(15 downto 0);
13    signal R : std_logic_vector(15 downto 0);
14    signal N, C, V : std_logic;
15    signal Nulo, Carry, V_over : std_logic;
16    -- registros de la parte de control
17    signal RI : std_logic_vector(15 downto 0);
18    signal PC : std_logic_vector(11 downto 0);
19    signal CC : std_logic;
20    -- partes de la instrucción
21    signal tipo : std_logic;
22    signal COD : std_logic_vector(3 downto 0);
23    signal DIR : std_logic_vector(11 downto 0);
24    -- señal de validación de dato
25    signal DATVint : std_logic;
26    -- ALU como componente
27    component ALU port( A, B : in std_logic_vector(15 downto 0);
28                        COD : in std_logic_vector(3 downto 0);
29                         CC : in std_logic;
30                          R : out std_logic_vector(15 downto 0);
31                          C : in std_logic;
32         Nulo, Carry, V_over : out std_logic );
33    end component;
34
35    begin
36    -- partes de una instrucción
37    tipo <= RI(15); COD <= RI(15 downto 12); dir <= RI(11 downto 0);
38    -- inserción de la ALU como componente
39    U1: ALU port map( A => A, B => B, COD => COD, CC => CC, R => R,
40    C => C, Nulo => Nulo, Carry => Carry, V_over => V_over );
41    -- BIESTABLES INDICADORES
```

Los indicadores solamente deben modificarse en determinadas instrucciones, tomando, en tales casos, el valor que ha sido generado en el componente **ALU**; en el resto de instrucciones el indicador debe conservar el valor que tenía previamente:

- **N** (resultado nulo) debe actualizarse en las instrucciones aritméticas y lógicas;

- **C** (acarreo) debe modificarse solamente en las instrucciones de suma y resta (**ADD**, **SUB**) y en la de desplazamiento (**SRR**);

- **V** (desbordamiento, *over-flow*) se modificará sólo en la suma y en la resta.

```
47    process(RS, CK)
48      begin
49        if RS = '1' then N <= '0'; C <= '0'; V <= '0';
50          elsif CK'event and CK = '1' then
51           if CC = '1' then -- ciclo de ejecución
52            -- indicador de resultado nulo: N todas las inst. de tipo a)
53               if tipo = '0' then N <= Nulo; end if;
54              -- indicador de acarreo: C inst. de suma, resta o desplazam.
55                if COD = "0000" or COD = "0001" or COD = "1111"
56                   then C <= Carry; end if;
57                if COD = "1010" then C <= '0'; end if; -- instrucción CLR (borrado)
58                -- indicador de desbordamiento: V sólo inst. de suma o resta
59                if COD = "0000" or COD = "0001" then V <= V_over; end if;
60          end if;
61      end if;
62    end process;
63  -- ACCESO A MEMORIA: manejo de los buses de direcciones y control
```

Cuando **CC = 0**, ciclo de búsqueda, debe producirse una lectura de la dirección de memoria señalada en el contador de programa **PC**; también debe producirse lectura de la memoria en el ciclo de ejecución de las instrucciones **LDA** y **LDB**, sobre la dirección **DIR** incluida en la instrucción.

Cuando **CC = 1**, ciclo de ejecución, debe producirse una escritura en la memoria, en la dirección **DIR**, en los siguientes casos: instrucciones aritméticas y lógicas, instrucción de borrado **CLR** e instrucción de almacenamiento del contador de programa **SPC**.

```
66      process(CC, tipo, COD, DIR, PC, R)
67      begin
68        R_W <= '1'; DATVint <= '0'; DIRV <= '0'; -- valores por defecto
69        if CC = '0' then Dirbus <= PC; DIRV <= '1'; -- búsqueda de instrucción
70        else Dirbus <= DIR; -- ejecución de instrucción
71          -- lectura de memoria: solo en las instrucciones LDA y LDB
72          if COD = "1000" or COD = "1001" then DIRV <= '1'; end if;
73          -- escritura en memoria
74          -- instrucciones de tipo a), instrucción CLR e instrucción SPC
75          if tipo = '0' or COD = "1010" or COD = "1011" then
76              R_W <= '0'; DATVint <= '1'; DIRV <= '1';
77          end if;
78        end if;
79      end process;
80      -- ESCRITURA: salida y validación de dato para una operación de escritura
```

El procesador debe «poner» un dato en el bus de datos en los siguientes casos:

- en las instrucciones aritméticas y lógicas debe enviar el resultado **R** ;
- en la instrucción de borrado **CLR** debe poner valor 0 (**0000000000000000)**;
- en la instrucción **SPC** debe enviar el valor del contador de programa **PC**.

En el resto de instrucciones el procesador debe dejar las líneas del bus de datos (de tipo *inout*) en alta impedancia.

En el caso de la instrucción **SPC**, como el contador de programa solamente utiliza 12 de los 16 bits, se completan los 4 iniciales con el código de la instrucción de salto incondicional **JMP**; esto resulta útil para el manejo de subrutinas (cuestión esta que dejamos abierta para consideración de posibles interesados).

La validación del dato **DATV** debe producirse cuando el dato se encuentre situado, en forma correcta y estable, en el bus de datos; por ello, la retrasamos hasta la segunda fase del ciclo de reloj (**CK = 0**) dando tiempo en la primera fase a que el procesador coloque el dato en el bus, se completen todos los transitorios de salida y el valor del bus de datos sea correcto y estable.

```
82      Databus <= R when tipo = '0' and CC = '1' else
83      "0000000000000000" when COD = "1010" and CC = '1' else
84      "1100" & PC when COD = "1011" and CC = '1' else
85      "ZZZZZZZZZZZZZZZZ";
86      -- validación de dato (retardada para asegurar el envío previo del dato)
87      DATV <= DATVint and (not CK);
88
89      -- MANEJO DE LOS REGISTROS: RI, CC, PC, A, B
```

El registro de instrucciones **RI** debe recibir el contenido del bus de datos en todos los ciclos de búsqueda de instrucción, es decir, siempre que **CC = 0**. El contador de ciclos debe cambiar de valor (cuenta módulo 2: **0**, **1**, **0**, **1**, **0**…) con cada ciclo de reloj, ya que se alternan los ciclos de búsqueda y de ejecución. El contador de programa **PC** debe pasar a señalar la posición de memoria siguiente al finalizar cada ciclo de búsqueda, en el cual ya ha utilizado la dirección que tenía.

El programa comienza su ejecución por la posición 0 del mapa de memoria, pues la inicialización *(Reset)* «borra» el contenido del contador de programa (asimismo, la inicialización pone a **0** el contador de ciclos, para comenzar con un ciclo de búsqueda). Los saltos «cargan» el contador de programa con la dirección **DIR** a la que señalan, con tal de que se cumpla la correspondiente condición del salto En el ciclo de ejecución (**CC = 1**), con la instrucción **LDA**, el acumulador **A** recibe un dato desde la memoria y, en las instrucciones aritméticas y lógicas, recoge el resultado; de igual modo, **B** recibe valor con la instrucción **LDB** y se borra con **CLR**.

```
92      process(RS, CK)
93      begin
94        if RS = '1' then
95            RI <= (others => '0');
96            CC <= '0'; PC <= (others => '0');
97            A <= (others => '0');
98            B <= (others => '0');
99        elsif CK'event and CK = '1' then
100       -- Registro de instrucciones
101         if CC = '0' then RI <= Databus; end if;
102       -- Contador de ciclos (biestable T, contador módulo 2)
103         CC <= not CC;
104       -- Contador de programa
105         if CC = '0' then PC <= PC + 1;
106         else if COD = "1100" then PC <= DIR; end if; -- saltos
107         if COD = "1101" and N = '1' then PC <= DIR; end if;
108         if COD = "1110" and C = '1' then PC <= DIR; end if;
109         if COD = "1111" and V = '1' then PC <= DIR; end if; end if;
110       -- Acumuladores
111         if CC = '1' then
112       -- operaciones de tipo a)
113         if tipo = '0' then A <= R; end if;
114       -- operaciones LDA, LDB y C LR
115         if COD = "1000" then A <= Databus; end if;
116       if COD = "1001" then B <= Databus; end if;
117       if COD = "1010" then B <= (others => '0'); end if; end if;
118       end if;
119     end process;
120   end PROCESADOR;
```