# UN MICROPROCESADOR ELEMENTAL COMO EJERCICIO DE DISEÑO DE SISTEMAS DIGITALES

T. POLLÁN, C. BERNAL Y A. BONO
*Escuela Universitaria de Ingeniería Técnica Industial de Zaragoza.*
*Departamento de Ingeniería Electrónica y Comunicaciones. Universidad de Zaragoza.*
tpollan@unizar.es

*El microprocesador es reconocido como la «cumbre» paradigmática de los sistemas digitales y es percibido como muy potente y complejo. Por ello, se propone un microprocesador elemental concreto como ejemplo útil de diseño digital de complejidad viable, es decir, de sistema de relativa complejidad pero abordable como ejercicio o trabajo de asignatura (más aún, si su descripción se hace a través de un lenguaje circuital, como puede ser el VHDL).*

.

## 1. Introducción

Es frecuente que, en congresos o reuniones sobre metodología docente, los profesores de electrónica digital nos recordemos la necesidad de que la enseñanza de esta materia no quede limitada al conocimiento de los bloques o "piezas" de diseño y al manejo de pequeños diagramas de bloques o a la descripción de sistemas muy simples. Entendemos que debemos aproximar al alumno al diseño de sistemas complejos, que son los sistemas digitales de autentico interés y utilidad hoy día.

Ciertamente, la disponibilidad de lenguajes de descripción circuital (en particular, VHDL y Verilog) ha permitido que, una vez conocidas las "piezas" de diseño digital (funciones booleanas, bloques combinacionales, grafos de estado, registros, contadores, …) podamos abordar el diseño y simulación de sistemas de diferentes grados de complejidad; además, disponemos de amplios dispositivos programables (CPLDs y FPGAs), relativamente baratos, donde implementar tales diseños.

Para ello, necesitamos enunciados de sistemas digitales que sean, a la vez, razonablemente complejos y adecuados como ejercicios de asignatura. Esta comunicación propone como ejercicio de diseño un microprocesador de 16 instrucciones con una configuración básica sencilla y ortogonal.

En el mismo sentido, en el anterior Congreso TAEE 2006, celebrado en la Universidad Politécnica de Madrid presentamos una comunicación con el título de "Máquinas algorítmicas como opción didáctica de sistemas digitales complejos" [1]. Propusimos como ejemplos útiles de diseño digital de complejidad viable aquellos sistemas que pueden plantearse como máquinas algorítmicas, con separación entre parte de control y parte operativa y descripción del control en forma de algoritmo, representable en un grafo de estados. Es decir, las máquinas algorítmicas son sistemas de relativa complejidad pero abordables como ejercicios o trabajos de asignatura (más aún, si su descripción se hace a través de un lenguaje circuital, como puede ser el VHDL); además, presentan características didácticas muy positivas.

Ahora bien, en buena medida, el microprocesador es reconocido como la «cumbre» paradigmática de los circuitos integrados digitales y es percibido como un sistema digital muy potente y, por lo mismo, sumamente complejo. Se trata, pues, de mostrar que, a pesar de tal complejidad, su diseño no presenta una dificultad especial y puede ser desarrollado, con relativa normalidad, a partir de un «recorrido» estructurado de su configuración y su funcionamiento.

Se trata, también, de desarrollar una sencilla aproximación a la arquitectura de computadores, esto es, a la configuración de los procesadores que actúan bajo programa. Para ello, consideraremos un pequeño «microprocesador» con un repertorio relativamente reducido de instrucciones y con un solo modo de direccionamiento: directo (para evitar la necesidad de introducir conceptos de «segundo orden», como el direccionamiento indexado o similares).

En la bibliografía existen numerosos procesadores con muy pocas instrucciones, que han sido presentados con el objetivo de servir como una primera, y muy básica, introducción a la arquitectura de computadores. Algunos manejan solo 4 instrucciones; por ejemplo, la «máquina sencilla» propuesta por M. Valero y E. Ayguadé (U. Politécnica de Cataluña.) [2] y [3] o el «procesador simple» de R. H. Katz *(Contemporary Logic Design)* [3].

El «microprocesador elemental» que aquí se presenta es deudor, en buena medida, de las ideas de los autores citados, pero su conjunto de instrucciones no es tan «mínimo»: se dispone de un mayor número de instrucciones, como aproximación más realista a los procesadores habituales.

Los microprocesadores de «muy pocas instrucciones» presentan el inconveniente de que son «poco realistas» y muy complejos de programar para la realización de cálculos o controles de verdadero interés; su programación requiere mucho «talento», pues es preciso utilizar, en forma ingeniosa, sus instrucciones para lograr acciones que no les corresponden directamente.

En el otro extremo, los microprocesadores comerciales o los «microprocesadores virtuales», diseñados para su programación en FPGAs (como puede ser el PICOBLAZE de Xilinx), ofrecen un conjunto tan amplio de instrucciones que requieren una amplia dedicación de tiempo para comprender y «memorizar» lo que cada instrucción hace. Y su diseño resultaría innecesariamente extenso, sobre todo para una primera aproximación, con el riesgo de que la diversidad de acciones programables dificulte la comprensión de la configuración circuital básica de un microprocesador y de su descripción estructural.

En un punto medio equidistante se sitúa nuestra propuesta de microprocesador de 16 instrucciones; las suficientes para abordar en forma directa cálculos y controles de utilidad, con relativa facilidad en la confección de sus programas, y en número no excesivo para comprender fácilmente la acción que cada una realiza y la estructura global del procesador resultante.

Cada instrucción tiene dos partes: su código de instrucción **COD** que ocupa los 4 bits más significativos y una dirección de memoria **DIR** (de 12 bits) sobre la que actúa la instrucción.

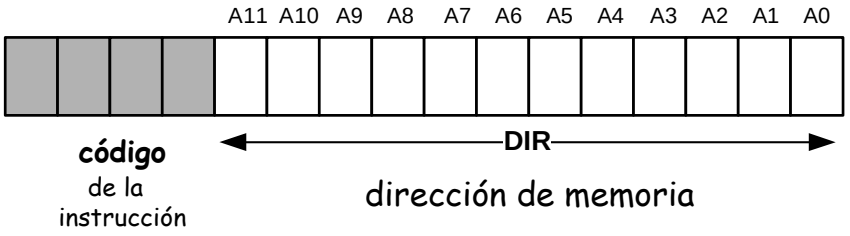


**Figura 1**. Formato de una instrucción

## 2. Configuración del microprocesador

El microprocesador utilizará una longitud de palabra de 16 bits, tanto para los datos como para las instrucciones. De forma que un dato numérico que ocupe una sola palabra tendrá una magnitud entre 0 y 65.535, si es entero positivo, y entre -32.768 y +32.767, si se admiten, también, los enteros negativos y se utiliza codificación en complemento a 2.

Dado que el código de instrucción ocupa 4 de esos bits (para diferenciar las 16 posibles) quedan 12 bits para el direccionamiento de dato: cada instrucción contendrá siempre una dirección de memoria, de forma que la anchura del bus de direcciones será de 12 líneas y 4 K ($2^{12}$) el tamaño del mapa de memoria.

Los terminales exteriores del procesador serán:

- Bus de datos de 16 líneas: datos e instrucciones de 16 bits
- Bus de direcciones de 12 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.

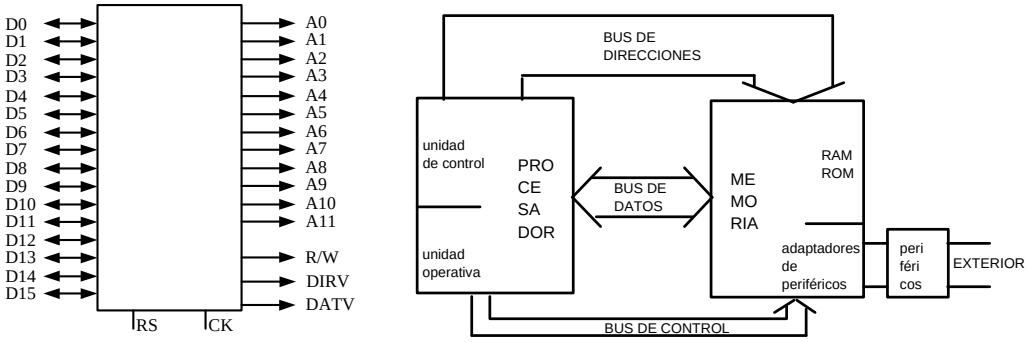


**Figura 2**. Terminales y diagrama de conexión del microprocesador elemental

Las 16 instrucciones de este microprocesador se dividen en 5 grupos:

- aritméticas: suma **ADD**, resta **SUB** y decrementar **DEC**;
- lógicas: operaciones "**y**" **AND** y "**o**" **ORA** e inversión **INV**;
- desplazamientos: a izquierda **SRL** y derecha **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 **BRN**, acarreo **BSC** y desbordamiento **BSV**.

Un procesador, como sistema digital con esquema de cálculo complejo puede ser dividido en una parte operativa y otra de control; tal diferenciación facilita su diseño, por cuanto permite tratar por separado las operaciones a realizar y la secuencia en que tales operaciones se realizan :

A) La parte operativa efectúa las operaciones y almacena operandos y resultados; contendrá dos registros acumuladores **A** y **B**, una **ALU** con capacidad de suma y resta y de operaciones lógicas, y tres biestables indicadores de resultado nulo **N**, acarreo **C** y desbordamiento **V**.

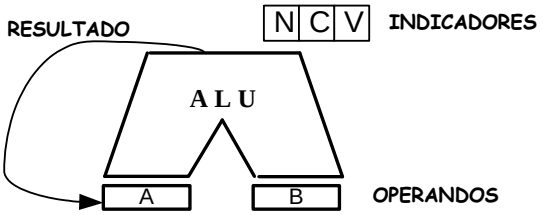


**Figura 3**. Parte operativa del microprocesador elemental

Los registros o acumuladores **A** y **B** actúan como operandos y el acumulador **A** recoge, también, el resultado de la operación; si la operación es de un solo operando utiliza solamente el acumulador **A**.

El indicador de acarreo **C** sirve para operaciones aritméticas con datos que ocupan más de una palabra y el de desbordamiento **V** *(over-flow)* permite supervisar la adecuación del resultado para números codificados en complemento a 2. Los indicadores de acarreo **C** y de resultado nulo **N** pueden ser utilizados como salidas de una comparación (menor, igual) por medio de una instrucción de resta (**SUB**). Existe una instrucción de salto condicionada a cada uno de los tres indicadores, es decir, a que el resultado sea nulo **N**, a la existencia de acarreo **C** y a la situación de desbordamiento **V**, respectivamente.

B) La unidad de control necesitará un contador de programa **PC** para señalar la dirección de la siguiente instrucción a ejecutar, un registro de instrucciones **RI** para almacenar la instrucción que se está ejecutando y el correspondiente circuito secuencial decodificador y controlador de la ejecución de las instrucciones.

Por simplicidad, se ha previsto que la aplicación de todas las instrucciones sea homogénea y ocupe, para cada una de ellas, dos ciclos de reloj: un primer ciclo de búsqueda, en que la instrucción es recogida desde la memoria y almacenada en el microprocesador, y un segundo ciclo de ejecución, en que la operación que prescribe es ejecutada y, al finalizar ese ciclo, el resultado es almacenado donde proceda.

Un biestable tipo T (contador mólulo 2) diferenciará, sucesivamente, esos dos ciclos de aplicación de cada instrucción: **CC** (contador de ciclos).

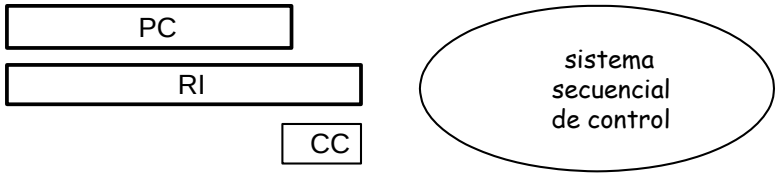


**Figura 4**. Parte de control del microprocesador elemental

## 2. Parte operativa: instrucciones aritméticas, lógicas y desplazamientos

Hay instrucciones que se aplican sobre dos operandos y utilizan como tales los acumuladores **A** y **B**; el resultado se almacena en **A** y, también, en la dirección de memoria indicada en la propia instrucción.

| | CÓDIGO | operación | Indicadores a los que afectan |
|---|---|---|---|
| **ADD** | **0000** | A **+** B | N (resultado nulo), C (acarreo) y V (desbordamiento) |
| **SUB** | **0001** | A **-** B | N (resultado nulo), C (acarreo) y V (desbordamiento) |
| **AND** | **0010** | A "**y**" B | N (resultado nulo) |
| **ORA** | **0011** | A "**o**" B | N (resultado nulo) |

**Tabla 1**. Instrucciones aritméticas y lógicas de dos operandos

Otras instrucciones se aplican sobre un solo operando, el acumulador **A**, operando y resultado, que, también, se almacena en la dirección de memoria **DIR** indicada en la instrucción.

| | CÓDIGO | operación | Indicadores a los que afectan |
|---|---|---|---|
| **DEC** | **0100** | A **-** 1 | N (resultado nulo) |
| **INV** | **0101** | $\overline{A}$ | N (resultado nulo) |
| **SRL** | **0110** | A $\leftarrow$ | N (resultado nulo), C(acarreo) |
| **SRR** | **0111** | A $\rightarrow$ | N (resultado nulo), C(acarreo) |

**Tabla 2**. Instrucciones aritméticas, lógicas y desplazamientos de un solo operando

Respecto a las 8 instrucciones anteriores: resultado $\rightarrow$ acumulador **A**
resultado $\rightarrow$ memoria(**DIR**).
Existirán ocasiones en que no sea de interés almacenar el resultado en memoria; en tal caso deberá indicarse en la instrucción la dirección de memoria **000000000000**.

Los operandos deberán encontrarse previamente en los acumuladores **A** y **B**, mediante instrucciones de carga (**LDA** y **LDB**) o como resultado de operaciones anteriores. 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 acumulador **A** y en la memoria (en la dirección **DIR** contenida en la propia instrucción); salvo que la dirección de memoria **DIR** sea 000000000000, en cuyo caso no hay acceso a memoria.

La instrucción **SRR** *(Shift Register Left)* realiza un desplazamiento del operando hacia la izquierda (un bit), poniendo el bit menos significativo a **0** y llevando el más significativo al indicador de acarreo **C**; **SRL** *(Shift Register Righ)* realiza un desplazamiento hacia la derecha, poniendo el bit más significativo a **0** y llevando el menos significativo (unidades) al indicador de acarreo **C**.

No existe instrucción específica (por limitación de número) para el incremento en una unidad, pero puede ser realizado ejecutando un decremento **DEC** precedido y seguido por una inversión **INV**: **INV DEC INV**. Caso de que interese programar un contador módulo N puede hacerse con la instrucción de decrementar **DEC**, cargando el número N-1 cada vez que el contador llegue a 0. La configuración de contadores decrecientes tiene la ventaja de que resulta directa la detección de su mínimo (valor 0) que se refleja directamente sobre el indicador N.

## 2.1. Descripción VHDL de la parte operativa: ALU

Las instrucciones anteriores determinan la parte operativa del microprocesador, es decir, todas las operaciones a realizar por la ALU del mismo; procede, pues, describir tal parte operativa en un módulo específico.

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

entity ALU is
      port ( A, B        : in   std_logic_vector (15 downto 0);   -- acumuladores
             COD         : in   std_logic_vector (3 downto 0);    -- código de instrucción
             CC          : in   std_logic;                        -- ciclo de aplicación
             R           : out std_logic_vector (15 downto 0);    -- resultado
             C           : in  std_logic;                         -- entrada de acarreo
             Nulo, Carry, V_over    : out std_logic  );           -- indicadores
end ALU;

architecture OPERACIONES of ALU is

-- necesidad de una señal de resultado de un bit más para detectar acarreos
signal RR         : STD_logic_vector(16 downto 0);

begin

-- operaciones de la ALU
process(COD, A, B, C, CC)
begin
if CC = '0' or COD(3) = '1' then RR <= (others => '0');
-- solo toman valor en el ciclo de ejecución de instrucciones de la parte operativa
      else
      RR(16) <= '0';            -- por defecto para operaciones que no utilizan acarreo
         case COD is
               when "0000" => RR <= '0' & A + B + C;          -- suma (con acarreo)
               when "0001" => RR <= '0' &  A - B - C;         -- resta (con acarreo)
               when "0010" => RR(15 downto 0) <= A and B;     -- and
               when "0011" => RR(15 downto 0) <= A or B;      -- or
               when "0100" => RR(15 downto 0) <= A - 1;       -- decrementar
               when "0101" => RR(15 downto 0)<= not A;        -- invertir
               when "0110" => RR <= A & '0';
                                    -- deplazamiento hacia la izquierda
               when "0111" => RR <= A(0) & '0' & A(15 downto 1);
                                    -- deplazamiento hacia la derecha
               when others  => RR <= (others => '0');
end case;
end if;
end process;

-- resultado
R <= RR(15 downto 0);
```

```
-- valor de los indicadores
process(COD,RR, A, B, CC)
begin
    -- resultado nulo
     if RR(15 downto 0) = "0000000000000000" then Nulo <= '1'; else Nulo <= '0'; end if;
    -- acarreo
    Carry <= RR(16);
    -- desbordamiento (over-flow)
    V_over <= '0';                    -- por defecto
    if COD = "0000" then                                          -- operación de suma
      if      (A(15) = '0' and B(15) = '0' and RR(15) = '1')
          or  (A(15) = '1' and B(15) = '1' and RR(15) = '0') then V_over<= '1'; end if;
    end if;
    if COD = "0001" then                                          -- operación de resta
      if      (A(15) = '0' and B(15) = '1' and RR(15) = '1')
          or  (A(15) = '1' and B(15) = '0' and RR(15) = '0') then V_over<= '1'; end if;
    end if;
end process;
end OPERACIONES;
```

## 3. Instrucciones de transferencia y de salto

### 3.1. 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**) $\rightarrow$ A | No afectan a ninguno |
| **LDB** | **1001** | memoria(**DIR**) $\rightarrow$ B | No afectan a ninguno |
| **CLR** | **1010** | 0 $\rightarrow$ memoria(**DIR**) ; 0 $\rightarrow$ B; 0 $\rightarrow$ C | C(acarreo) |
| **SPC** | **1011** | PC + 2 $\rightarrow$ memoria(**DIR**) | No afectan a ninguno |

**Tabla 3**. Instrucciones de transferencia con memoria

Como todas las demás, estas instrucciones se completan en dos ciclos de reloj, 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.

La instrucción **SPC** permite utilizar subrutinas: su configuración es un poco compleja pero sirve para remarcar la actuación estructural de tales subrutinas. Para que un segmento de un programa actúe como subrutina, bastará transferir la dirección actual del contador de programa al final de dicho segmento mediante una instrucción **SPC** y efectuar, inmediatamente después, un salto a la dirección de comienzo de la subrutina. La instrucción **SPC** guarda en memoria, en los cuatro bits más significativos, el código correspondiente a una instrucción de salto no condicionada y, en los bits siguientes, el contenido del contador de programa que señalaba a la instrucción **SPC** aumentado en dos unidades (para dejar sitio, en medio, a la instrucción de salto que ejecuta la subrutina):

| | | |
|---|---|---|
| *n* | SPC DIR | paso del contador de programa (n+2) al final de la subrutina |
| *n+1* | JMP DIR | salto al comienzo de la subrutina |
| *n+2* | | dirección de vuelta de la subrutina |

## 3.2. 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 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** $\rightarrow$ PC (incondicional) | Ninguno |
| **BRN** | **1101** | **DIR** $\rightarrow$ PC si N = **1** | Ninguno |
| **BRC** | **1110** | **DIR** $\rightarrow$ PC si C = **1** | Ninguno |
| **BRV** | **1111** | **DIR** $\rightarrow$ PC si V = **1** | Ninguno |

**Tabla 4**. Instrucciones de salto

En el segundo ciclo de reloj, la primera instrucción transfiere el contenido de la posición de memoria **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 **BRN** se produce si el resultado de la operación anterior es nulo (**N = 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 **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.

## 4. Descipción VHDL de la parte de control

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

entity MICRO is
        port (  RS, CK                  : in     std_logic;
                Databus                 : inout std_logic_vector(15 downto 0);
                Dirbus                  : out    std_logic_vector(11 downto 0);
                R_W, DIRV, DATV         : out    std_logic  );
end MICRO;

architecture PROCESADOR of MICRO is
    -- acumuladores, resultado e indicadores
    signal A, B                     : std_logic_vector(15 downto 0);
    signal R                        : std_logic_vector(15 downto 0);
    signal N, C, V                  : std_logic;
    signal Nulo, Carry, V_over      : std_logic;
    -- registros de la parte de control
    signal RI           : std_logic_vector(15 downto 0);
    signal PC           : std_logic_vector(11 downto 0);
    signal CC           : std_logic;
    -- partes de la instrucción
    signal COD          : std_logic_vector(3 downto 0);
    signal DIR          : std_logic_vector(11 downto 0);
    -- señal de validación de dato
    signal DATVint      : std_logic;
    -- ALU como componente
    component ALU port(  A, B                   : in   std_logic_vector(15 downto 0);
                         COD                    : in   std_logic_vector(3 downto 0);
                         CC                     : in   std_logic;
                         R                      : out std_logic_vector(15 downto 0);
                         C                      : in  std_logic;
                         Nulo, Carry, V_over    : out std_logic   );
    end component;
begin
-- partes de una instrucción
COD <= RI(15 downto 12);     DIR <= RI(11 downto 0);
-- inserción de la ALU como componente
U1: ALU port map(   A => A, B => B, COD => COD, CC => CC, R => R,
                    C => C, Nulo => Nulo, Carry => Carry, V_over => V_over  );
```

## 4.1. 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 de la parte operativa, borrado **CLR** y almacenamiento del contador de programa **SPC**; el acceso a memoria no se producirá si **DIR** = 000000000000.

```
process(CC, COD, DIR, PC, R)      begin
R_W <= '1';            DATVint <= '0';        DIRV <= '0';   -- valores por defecto
if CC = '0' then       Dirbus <= PC;          DIRV <= '1';    -- búsqueda de instrucción
      else             Dirbus <= DIR;                         -- ejecución de instrucción
            -- lectura de memoria: solo en las instrucciones LDA y LDB
            if COD = "1000" or COD = "1001" then    DIRV <= '1';     end if;
            -- escritura en memoria
            -- instrucciones de la parte operativa, instrucción CLR  e  instrucción SPC
            if COD(3) = '0' or COD = "1010" or COD = "1011" then
                  if DIR /= 0 then
                        R_W <= '0';     DATVint <= '1';    DIRV <= '1';   end if;
            end if;
end if;     end process;
```

## 4.2. 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** (PC + 2).

En el resto de instrucciones 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 en el manejo de subrutinas (según se ha comentado al introducir dicha instrucción), para lo cual, además, se incrementa previamente el contador de programa en 2 unidades.

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 de los buses de direcciones y de datos sean correctos y estables.

```
Databus <=  R                    when COD(3) = '0' and CC = '1'     else
            "0000000000000000"   when COD = "1010" and CC = '1'     else
            "1100" & PC          when COD = "1011" and CC = '1'     else
            "ZZZZZZZZZZZZZZZZ";

-- validación de dato (retardada para asegurar el envío previo del dato)
DATV <= DATVint and (not CK);
```

## 4.3. 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 (contaje módulo 2: **0**, **1**, **0**, **1**, **0**…) en cada ciclo de reloj, alternando 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; en el caso de la instrucción **SPC** el contador de programa debe aumentar en dos unidades.

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 de la parte operativa, recoge el resultado; de igual modo, **B** recibe valor con la instrucción **LDB** y se borra con **CLR**.

```
process(RS, CK)       begin
if RS = '1' then  RI <= (others => '0');   CC <= '0';  PC <= (others => '0');
                  A <= (others => '0');   B <= (others => '0');
    elsif CK'event and CK = '1' then
-- Registro de instrucciones
       if CC = '0' then RI <= Databus; end if;
-- Contador de ciclos (biestable T, contador módulo 2)
       CC <= not CC;
-- Contador de prograna
       if CC = '0' then PC <= PC + 1;
                        if COD = "1011" then PC <= PC + 2;  end if;            -- "subrutinas"
           else   if COD = "1100"                then   PC <= DIR;  end if;    -- saltos
                  if COD = "1101" and N = '1'    then   PC <= DIR;  end if;
                  if COD = "1110" and C = '1'    then   PC <= DIR;  end if;
                  if COD = "1111" and V = '1'    then   PC <= DIR;  end if;        end if;
-- Acumuladores
       if CC = '1' then
            -- instrucciones de la pare operativa
            if COD(3) = '0'                then   A <= R;                 end if;
            -- operaciones LDA, LDB y C LR
            if COD = "1000"                then   A <= Databus;           end if;
            if COD = "1001"                then   B <= Databus;           end if;
            if COD = "1010"                then   B <= (others => '0');   end if;    end if;
end if;
end process;
```

4.4. 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 de la parte operativa;
- **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.

```
process(RS, CK)       begin
if RS = '1' then  N <= '0';  C <= '0';  V <= '0';
   elsif CK'event and CK = '1' then
      if CC = '1' then                  -- toman valor en el ciclo de ejecución
         -- indicador de resultado nulo: N       todas las inst. de la parte operativa
         if COD(3) = '0'                          then   N <= Nulo;       end if;
         -- indicador de acarreo: C              inst. de suma, resta o desplazam.
         if COD = "0000" or COD = "0001" or COD = "0110" or COD = "0111"
                                                  then   C <= Carry;      end if;
         if COD = "1010"  then C <= '0'; end if;          -- instrucción CLR (borrado)
         -- indicador de desbordamiento: V     sólo inst. de suma o resta
         if COD = "0000" or COD = "0001"          then   V <= V_over;    end if;
         end if;
end if;
end process;
end PROCESADOR;
```

## Referencias


[1] T. Pollán, B. Martín y J. Ponce de León. *Máquinas algorítmicas como opción didáctica de sistemas digitales complejos.* Actas del VII Congreso de Tecnologías Aplicadas a la enseñanza de la Electrónica. TAEE 2006.

[2] M. Valero y E. Ayguadé *La Máquina Sencilla: Introducción a la Estructura Bàsica de un Computador.* Publicació docent. Departament d'Arquitectura i Tecnologia de Computadors, UPC. (2001).

[3] J. García Zubía y otros. *Sistemas digitales y tecnología de computadores.* 2ª edición. Thompon (2007)

[4] R. H. Katz. *Contemporary logic design*. The Benjamín/Cummings Publishing Company, INC. (1994)

[5] T. Pollán. *Electrónica Digital. III. Microelectrónica*. Prensas Universitarias de Zaragoza (2007).