8 Unidad de proceso general

8.1 Introducción

En el capítulo anterior hemos diseñado procesadores de propósito específico (PPE): tanto la unidad de proceso (UP) como la de control (UC) eran diferentes para cada procesador según el problema que debía resolver (sumar m números, calcular el MCD, etc.). Las UP que hemos diseñado consisten en uno o varios registros interconectados entre sí a través de bloques combinacionales que procesan los datos (sumadores, restadores, comparadores, etc.). La interconexión entre estos elementos se controla, en algunos casos, mediante multiplexores de buses. La UC genera, en cada ciclo, las señales que controlan estos multiplexores para encaminar los datos, además de las señales de control de carga de los registros y las posibles señales que controlan la funcionalidad de algunos circuitos combinacionales.

En este capítulo seguiremos diseñando procesadores de propósito específico, pero ahora todos los procesadores, independientemente del problema que resuelvan, usarán una misma UP, que será de propósito general. Cambiar la funcionalidad del procesador consistirá en cambiar solamente la UC, que seguirá siendo de propósito específico.

A la vista de los diseños de las UP de propósito específico que hemos realizado en el capítulo anterior podemos decir que una UP general, que sirva para resolver cualquier problema, debe tener un conjunto de registros y un conjunto de circuitos combinacionales que realizan cálculos. Además, la interconexión entre los elementos se debe poder modificar dinámicamente, en cada ciclo, mediante multiplexores de buses.

En este capítulo diseñamos una UP general concreta, que denominamos UPG (Unidad de Proceso General), y que será el núcleo del computador que estamos construyendo. Todas las unidades funcionales que realizan cálculos las vamos a agrupar en un bloque combinacional que se denomina unidad aritmético-lógica, ALU (*Arithmetic-Logic Unit*) y todos los registros de la UPG se agrupan en un bloque secuencial que se denomina banco de registros, REGFILE (*Register File*). Estos dos grandes bloques se interconectan para permitir que en un ciclo se puedan leer dos registros del banco de registros, se puedan operar sus contenidos en la ALU y al final del ciclo se pueda escribir el resultado de la operación en un registro. Un esquema simplificado se muestra en la figura 8.1[1].

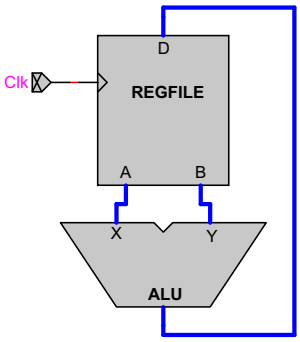


Fig. 8.1 Esquema simple de interconexión del banco de registros y la ALU de la UPG

Veamos primero (en las secciones 8.2 y 8.3) las funcionalidades y los circuitos internos de los bloques ALU y REGFILE, para luego (sección 8.4) interconectarlos y formar la UPG (añadiéndole, además, la capacidad de entrada y salida de datos). Después, detallamos las distintas acciones que se pueden realizar en la UPG en cada ciclo (sección 8.5) y diseñamos algunos PPE, como los del capítulo anterior,

1. La estructura de este diseño se puede considerar una generalización de la UP específica para sumar 4 números que vimos en en capítulo anterior, cambiando el registro por un banco de registros y el sumador por una ALU, aunque aquí de momento no hemos dibujado ningún bus de entrada de operandos ni de salida de resultados

pero usando la UPG (secciones 8.6 y 8.7). Ahora, para diseñar un PPE simplemente debemos especificar el grafo de estados de la UC, ya que la UPG no cambia nunca.

8.2 Unidad aritmético-lógica

La ALU es un circuito combinacional, en función del valor de los bits de los buses de control OP (de 2 bits) y F (de 3 bits) realizará una operación con los bits que se encuentran en los buses de datos de entrada X e Y (de 16 bits cada uno) y, pasado el tiempo de propagación correspondiente, estabilizará en el bus de salida W (de 16 bits) el resultado de la operación. Además, generará un 1 o un 0 en la señal binaria de salida z, indicando si los 16 bits del resultado presente en W son todos 0 o no.

En la figura 8.2 se muestra la ficha de la ALU que contiene un texto y una tabla que resume su funcionamiento, el símbolo del dispositivo con las señales y buses de entrada y de salida y su construcción interna usando los bloques funcionales AL (para realizar las operaciones aritméticas y lógicas cuando OP vale 00), CMP (para realizar comparaciones cuando OP vale 01) y MISC (para realizar otras operaciones variadas -misceláneas- cuando OP vale 10). La construcción interna de estos bloques, en función de otros dispositivos más elementales, se muestra en el apéndice de este capítulo junto con el bloque z que indica si el resultado es cero.

Dado que la ALU tiene 5 bits de control (2 de OP y 3 de F) para indicar la función que debe realizar, es posible codificar hasta 32 operaciones diferentes. No obstante, de momento no se usan todas estas posibilidades. La codificación OP=11 de momento no se usa (para no dejar la entrada 3 del multiplexor al aire se ha conectado a la entrada 2). Los 3 bits del bus F codifican qué operación aritmético-lógica (de las 8 implementadas que hay), de comparación (de las 5 que hay) o miscelánea (de las 2 que hay implementadas de momento) se debe efectuar en cada caso. De hecho, para simplificar la implementación, todas las operaciones se realizan en paralelo, pero sólo se muestra en la salida W el resultado de la operación seleccionada por los buses de control OP y F.

8.3 Banco de registros

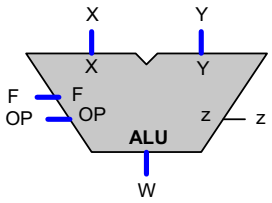
En la figura 8.3 se muestra la ficha del banco de registros que contiene un texto que resume su funcionamiento, el símbolo del dispositivo con las señales y buses de entrada y de salida (tanto de datos como de control) y el esquema lógico que muestra cómo está construido internamente usando bloques ya conocidos de capítulos anteriores. El banco de registros está formado por un conjunto de 8 registros de 16 bits cada uno, denominados R0, R1... R7. Tiene dos buses de datos de salida (de lectura), A y B, y uno de entrada (de escritura), D. El resto de las señales y buses son de control. Esto permite, durante un ciclo, leer dos registros y presentar su contenido en los buses A y B, y al final del ciclo, cuando llega el flanco ascendente de la señal de reloj (Clk), escribir los bits presentes en el bus D en un registro concreto, siempre y cuando la señal de permiso de escritura (WrD) valga 1.

En cada ciclo hay que indicarle al banco de registros qué registros concretos hay que leer y cuál hay que escribir al final del ciclo. Para ello se dispone de 3 buses de entrada de 3 bits cada uno (ya que se necesitan 3 bits para diferenciar los 8 registros): @A, @B y @D. Para leer por el bus A el contenido del registro Ri hay que codificar el valor i en los tres bits del bus @A. Pasado el tiempo de propagación necesario desde que son estables los tres bits de @A, quedará estable en el bus A el contenido del

16 Bit-wide ALU **ALU**

Description:

The 16-bit output W is the result of an arithmetic/logic operation (AL), a comparison (CMP) or an other action (MISC), applied to the 16-bit inputs X and Y. The type of operation is selected by the 2-bit input OP and by the 3-bit input F.



F	OP = 11	OP = 10	OP = 01	OP = 00
000	---	X	CMPLT(X, Y)	AND(X, Y)
001	---	Y	CMPLE(X, Y)	OR(X, Y)
010	---	---	---	XOR(X,Y)
011	---	---	CMPEQ(X, Y)	NOT(X)
100	---	---	CMPLTU(X, Y)	ADD(X, Y)
101	---	---	CMPLEU(X, Y)	SUB(X, Y)
110	---	---	---	SHA(X ,Y)
111	---	---	---	SHL(X, Y)

If the ALU performs a comparison the result can be true or false. If the result is true then W(b0) = 1 else W(b0) = 0. Moreover, W(bi) = 0 for i = 1 to 15.
The 1-bit output z indicates when the output W is a vector of 16 zeroes.

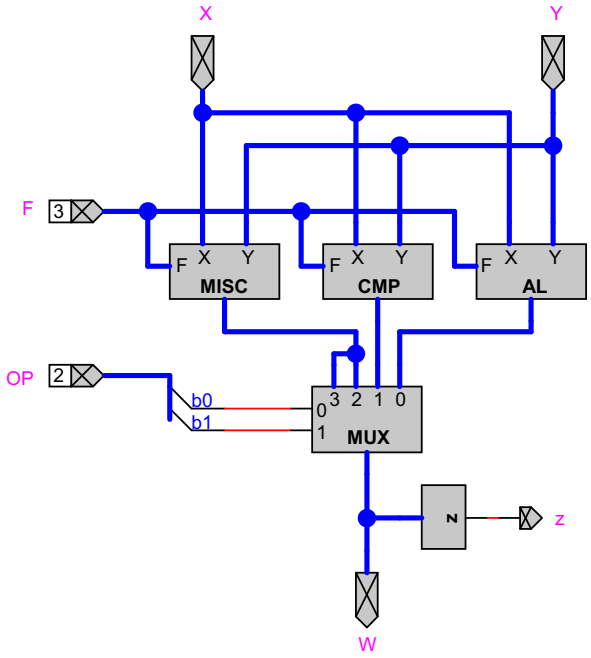


Fig. 8.2 Descripción, símbolo y realización interna de la unidad aritmético-lógica (ALU)

registro. Lo mismo para @B con relación a B. Para escribir el registro Rj con el valor codificado en el bus D deben permanecer estables las señales del bus D, las del bus @D (que codifica el valor j) y la señal binaria WrD (que debe valer 1) un tiempo antes de que llegue el flanco ascendente de reloj, que es cuando se escribe el registro. Este tiempo es el tiempo de propagación de estas señales desde su entrada hasta que estas señales llegan a la entrada de los biestables de los registros. En el ciclo en que la señal de permiso de escritura (WrD) vale 0 no se escribe ningún registro al final del ciclo.

Una vez entendido con profundidad lo que hace el banco de registros y la ALU y cómo lo hacen pasamos a formar la UPG conectando estos dos grandes bloques.

8.4 Unidad de Proceso General (UPG)

8.4.1 Estructura de la UPG

La figura 8.4 muestra el esquema lógico interno de la Unidad de Proceso General (UPG). Está formada por un banco de registros y una ALU, como los que acabamos de ver, interconectados formando un bucle, para que en cada ciclo de reloj se pueda operar el contenido de dos registros fuente y dejar el resultado, al final del ciclo, en un registro destino (que puede ser el mismo que uno de los registros fuente), como ya se mostró en el esquema simplificado de la figura 8.1.

Además, se ha dispuesto la mínima, pero necesaria, capacidad de comunicación con el exterior: un bus de entrada de datos desde el exterior al banco de registros (RD-IN, *read input*) y otro de salida del banco de registros, en concreto del bus A, al exterior (WR-OUT, *write output*). Hemos añadido un multiplexor

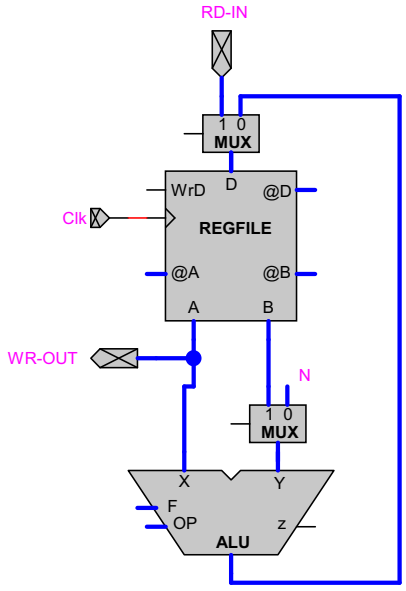


Fig. 8.4 Unidad de Proceso General, UPG sin señales de control

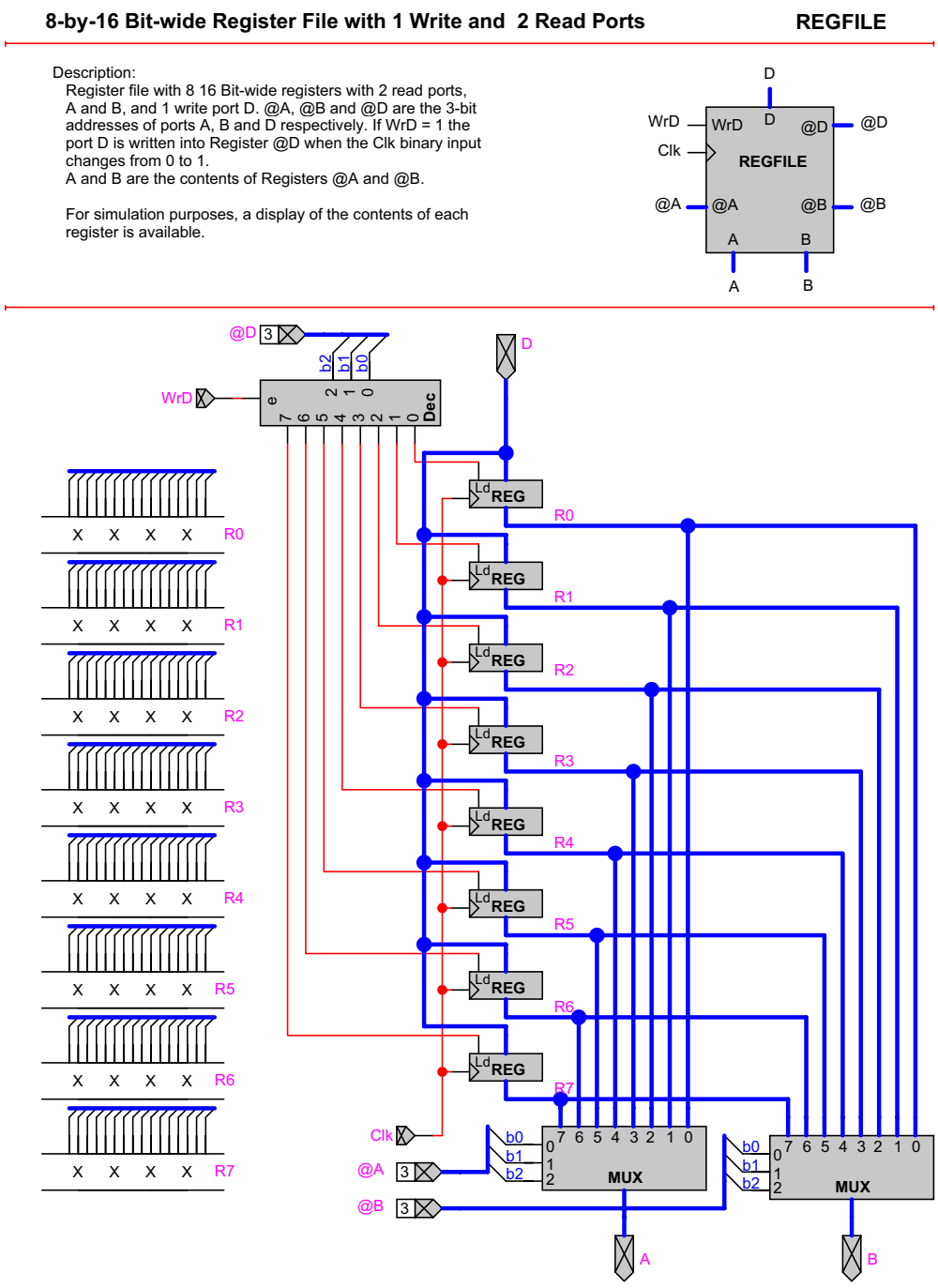


Fig. 8.3 Descripción, símbolo y realización interna del banco de registros (Register File)

de buses para elegir en cada ciclo qué se conecta al bus de escritura del banco de registros: el bus de entrada de datos o la salida de la ALU. En el capítulo siguiente veremos en detalle cómo se pueden conectar estos buses de entrada/salida de la UPG con un amplio conjunto de dispositivos externos al procesador.

También se ha dispuesto un multiplexor de buses en la entrada derecha de la ALU para seleccionar, en cada ciclo, si el operando Y de la ALU es el contenido de un registro del banco (que se lee por el bus B) o un valor constante de 16 bits, denominado N, que proporciona la UC formando parte de la palabra de control.

La UPG es de 16 bits, lo que quiere decir que almacena y opera datos de 16 bits. Por eso el ancho de los buses RD-IN, N y WR-OUT, el tamaño de los registros (ancho de los buses D, A y B) y el de la ALU (buses X, Y, W) es de 16 bits.

8.4.2 Conexionado entre la UPG y la unidad de control

La figura 8.5 muestra el esquema interno de un procesador de propósito específico formado por una unidad de control de propósito específico y la UPG. Las señales Control-In y Control-Out así como la UC dependerán de la funcionalidad del procesador, mientras que la UPG y su señal de salida (z) junto con sus buses de entrada (Palabra de control y RD-IN) y el de salida (WR-OUT) son siempre los mismos para cualquier procesador. La unidad de control (UC) debe generar en cada ciclo el valor concreto de los bits de la palabra de control para indicarle a la UPG qué debe hacer durante ese ciclo.

La figura 8.6 muestra la UPG indicando los nombres de cada una de las señales y buses (de 2, 3 y 16 bits) que forma la palabra de control (que va de la UC a la UPG), además del bit de condición z (que va de la ALU a la UC) y los buses de entrada (RD-IN) y de salida (WR-OUT) de datos del procesador.

Palabra de control. De momento, la palabra de control está formada por 33 bits (en el capítulo siguiente se completa al aumentar la capacidad de entrada/salida):

- 3 bits de cada uno de los 3 buses de dirección del banco de registros @A, @B y @D,
- los 2 bits de OP y los 3 de F que indican lo que debe hacer la ALU,

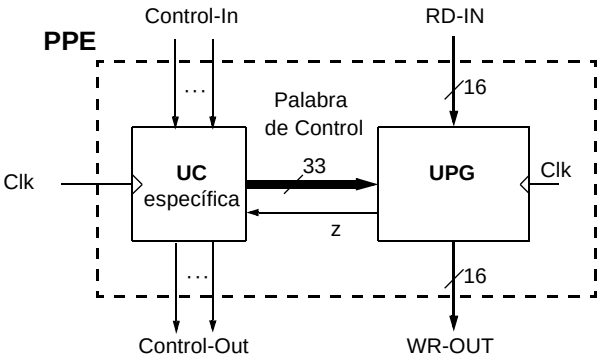


Fig. 8.5 Esquema general de interconexión de la UC con la UPG

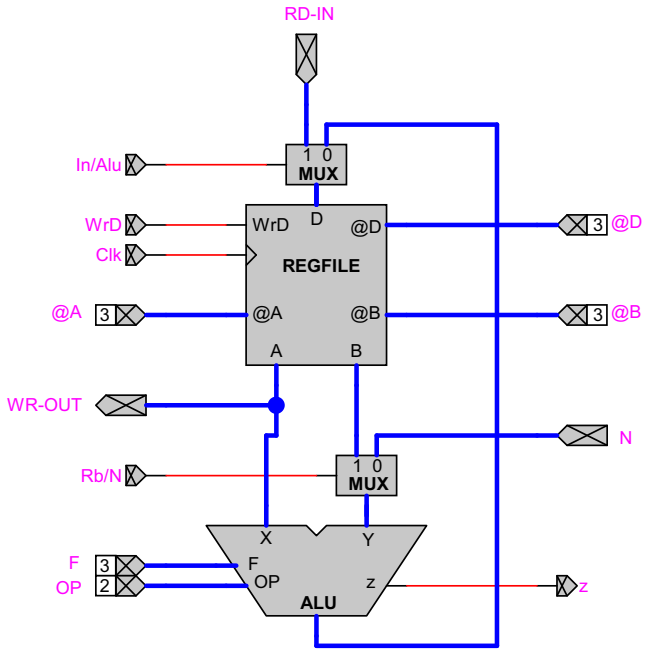


Fig. 8.6 Unidad de Proceso General, UPG

- los 16 bits del bus N,
- la señal de permiso de escritura WrD y
- las señales de selección de los multiplexores para encaminar los datos, In/Alu y Rb/N.

A continuación se muestran los 33 bits de la palabra de control, tal como los ordenaremos usualmente, aunque cualquier orden sería igualmente válido. Los 16 bits de N se han codificado en 4 dígitos hexadecimales (h_3, h_2, h_1, h_0) para reducir el espacio que ocupa la palabra de control en las figuras, y por comodidad al escribir palabras de control.

@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (hexa)			
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	X	X	X	X

Bit de condición. De la UPG sólo sale un bit, z, que le indica a la UC si el resultado que sale de la ALU (resultado de una comparación, o de una operación aritmética, lógica o de movimiento) es igual a cero. La UC puede usar el valor de z para tomar una decisión de cuál será el estado siguiente en su grafo de estados (cuál será la acción a realizar en la UPG en el ciclo siguiente).

8.5 Acciones en la UPG

En la figura 8.7 se muestra la tabla de codificación de las funciones de la ALU, que nos será útil en esta sección. Para obtener la descripción detallada de cada una de las operaciones aritméticas y lógicas de la ALU, ver el Anexo de este capítulo.

F			OP			
b_2	b_1	b_0	11	10	01	00
0	0	0	-	X	CMPLT (X, Y)	AND (X, Y)
0	0	1	-	Y	CMPLE (X, Y)	OR (X, Y)
0	1	0	-	-	---	XOR(X, Y)
0	1	1	-	-	CMPEQ (X, Y)	NOT (X)
1	0	0	-	-	CMPLTU (X, Y)	ADD (X, Y)
1	0	1	-	-	CMPLEU (X, Y)	SUB (X, Y)
1	1	0	-	-	---	SHA(X, Y)
1	1	1	-	-	---	SHL(X, Y)

Fig. 8.7 Funcionalidad de la ALU

Tipos de acciones. En la UPG se puede llevar a cabo, en cada ciclo, una de las siguientes acciones, que ordenamos por tipos de funcionalidad:

1. **Aritmético-lógicas y de comparación con dos operandos**. Leer dos registros, operarlos con alguna de las 12 funciones de 2 operandos que puede realizar la ALU y dejar el resultado en otro registro al final del ciclo. Las operaciones de dos operandos son: dos aritméticas, `ADD` y `SUB`; un desplazamiento aritmético, `SHA`, y otro lógico, `SHL`; 3 operaciones lógicas bit a bit, `AND`, `OR` y `XOR`; 2 de comparación de enteros, `CMPLT` y `CMPLE`, 2 de comparación de naturales, `CMPLTU` y `CMPLEU` y una comparación de igualdad `CMPEQ`. Un ejemplo de acción de la UP durante un ciclo consiste en escribir en el registro 6 la suma del contenido del registro 3 y del 5. Esta acción la denotaremos como:

   ```
   ADD R6, R3, R5
   ```

   (`ADD` es un mnemotécnico de la palabra inglesa *addition*, suma. Con `Ri` nos referimos al registro i del banco de registros).

   Otro ejemplo es: escribir en el registro 3, `R3`, un 1 (que es como codificamos el valor booleano `TRUE`, verdadero) si el contenido de `R5` es mayor o igual que el contenido de `R1`, interpretados los contenidos de los registros `R5` y `R1` como números naturales codificados en binario con 16 bits, y escribir un 0 (`FALSE`) en caso contrario. Como no existe la comparación de mayor o igual, podemos usar la complementada, cambiando el orden de los operandos, comparando si el contenido de `R1` es menor que el contenido de `R5`. Esta operación la denotamos como:

   ```
   CMPLEU    R3, R1, R5
   ```

   (`CMPLEU` es el mnemotécnico de *Comparison on Less or Equal for Unsigned*, comparación si menor o igual sin signo).

   En todos estos casos, el segundo operando puede ser, en vez del contenido de un registro, el número codificado en complemento a 2 con 16 bits que entra por `N`. Cuando el segundo operando es un

valor constante generado por la unidad de control en el campo N, terminaremos el mnemotécnico de la operación con la letra I (*Immediate*, Inmediato) y en vez del indicador del registro pondremos el valor del número que genera la unidad de control (expresado en decimal si no se indica lo contrario o en hexadecimal si se precede el número por 0x). Estos 16 bits los proporcionará la Unidad de Control. Dos ejemplos de operación con inmediato a nivel de mnemotécnico son:

```
ADDI  R7, R1, -1
ANDI  R2, R3, 0xFF00
```

2. **Aritmético-lógicas de un operando**. Sólo hay una acción: leer un registro por el bus A, operarlo con la función NOT (que es la única con un operando de las que realiza el bloque AL) y dejar el resultado en otro registro al final del ciclo. Por ejemplo, escribir en R4 el complemento lógico de cada bit del contenido de R2. Esta operación la denotaremos como:
   ```
   NOT  R4, R2
   ```
3. **De movimiento**. Veamos dos ejemplos. El primero consiste en copiar el contenido de un registro en otro registro. En el segundo ejemplo se trata de cargar un registro con un valor constante proporcionado por la UC a través del bus N. En este caso la ALU deja pasar la entrada Y a su salida W (OP=10 y F=001). Estas acciones las denominamos de movimiento de registro, el primer ejemplo, y de movimiento inmediato el segundo. Dos ejemplos como los comentados son respectivamente:
   ```
   MOVE  R1, R5
   MOVEI R3, 0xFA02
   ```
   (MOVE es un mnemotécnico de *move*, mover y la I final significa *immediate*, inmediato)
4. **De entrada de datos**. Escribir un registro, al final del ciclo, con la información presente durante este ciclo en el bus de entrada RD-IN. Por ejemplo, cargar R2 con el valor de la entrada de datos. Esto lo denotamos como:
   ```
   IN  R2
   ```
   (IN es un mnemotécnico de *input*, entrada).
5. **De salida de datos**. Que el contenido de un registro esté presente en el bus de salida WR-OUT durante este ciclo. Por ejemplo, sacar el contenido de R4. Esto lo denotamos como:
   ```
   OUT  R4
   ```
   (OUT es un mnemotécnico de *output*, salida).

8.5.1 Palabra de control para cada acción

Valor de cada bit de la palabra de control para un ejemplo de acción (`NOT R4, R2`). Para que la UPG haga una de estas acciones en un ciclo determinado, la UC debe proporcionar a la UPG la palabra de control adecuada durante ese ciclo. En la figura 8.8 se indica el valor concreto de cada bit (0, 1 o x) de la palabra de control para que la UPG realice la acción `NOT R4, R2`. Comentamos en detalle cada uno de los bits (en un orden distinto al que están escritos) justificando su valor. Para escribir, al final del ciclo, el registro R4 con el resultado de aplicar la operación lógica Not bit a bit a los 16 bits contenidos en el registro R2 se debe:.

- Leer por el bus A del banco de registros el contenido de R2 (ya que la ALU hace la operación NOT sobre el vector de 16 bits que le llega por su entrada X, que está directamente conectada a la salida A del banco de registros), para ello se codifica un 2 en el campo @A.

Palabra de Control de 33 bits

@A			@B			Rb/N	OP		F			In/Alu	@D			WrD	N (hexa)			
0	1	0	x	x	x	x	0	0	0	1	1	0	1	0	0	1	X	X	X	X

Fig. 8.8 Palabra de control de 33 bits para realizar la acción `NOT R4, R2`

- Poner los dos bits b_1b_0 de OP a 00 y poner 011 en el campo F, bits $b_2b_1b_0$, para que la ALU realice la `NOT` de su entrada X (ver tabla de funcionalidades de la ALU en la figura 8.7).
- Indicar al multiplexor de la parte superior de la UPG que deje pasar las señales que vienen de la ALU, codificando un 0 en el campo In/Alu, para dejar pasar el resultado de la ALU al bus de escritura del banco de registros.
- Poner la señal WrD a 1 para que al final del ciclo se escriba el resultado de la operación que se encuentra en el bus D en el registro que indican los bits del campo @D.
- Codificar un 4 en el campo @D para que el registro que se escriba al final del ciclo sea el `R4`.
- Codificar los campos Rb/N, @B y N de cualquier forma, pues es indiferente lo que entre por el bus B de la ALU, ya que la ALU no lo usa para nada (en estos casos pondremos una x, que en la implementación final puede ser tanto un 0 como un 1).

Valores de los bits de la palabra de control para los ejemplos tratados. La figura 8.9 muestra el valor de los 33 bits de la palabra de control para cada uno de los ejemplos anteriores (en las 9 primeras filas, con la fila de la `NOT` en gris) y para los ejemplos que comentamos en las tres siguientes subsecciones (3 últimas filas de la tabla). Pero antes una aclaración sobre la implementación de la acción `MOVE R1, R5`. Se puede hacer de dos formas: leer el registro por el bus A y hacer que la ALU conecte su bus X a su salida W (haciendo OP=10 y F=000) o bien leer el registro por el bus B, que el MUX-2-1 conecte su entrada 1 a su salida (haciendo Rb/N=1) y hacer que la ALU deje pasar el bus Y a su salida (OP=10 y F=001). La palabra de control de la tabla para este ejemplo usa la primera alternativa.

Acciones que no modifican los registros. Puede tener interés usar algunas de las acciones de la ALU sin escribir el resultado en un registro destino. Para indicarlo, en el campo del registro destino del mnemotécnico pondremos un guion. Esto será así cuando lo único que deseemos sea saber si el resultado de la acción es cero o distinto de cero observando el bit z que sale de la ALU. Por ejemplo,

```
ANDI  -, R3, 0x8000
```

le indica a la unidad de control que el bit 15 de `R3` vale 0 si la señal z vale 1, pero no modifica ningún registro del banco de registros de la UPG. En este caso, el campo WrD debe ser 0 y el campo @D y la señal In/Alu pueden tomar cualquier valor. La palabra de control completa se muestra en la antepenúltima fila de la tabla de la figura 8.9.

Acción `NOP`. Puede tener interés una acción que no haga nada (que pierda un ciclo) o hablando en positivo: que cuente un ciclo. Usamos el mnemotécnico `NOP` (*No OPeration*) para esta acción. Puede ser usada, por ejemplo, para sincronizar la entrada de datos si se deben leer dos datos que se encuentran

	Palabra de Control de 33 bits								
Mnemotécnicos	@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N (hexa)
ADD R6, R3, R5	0 1 1	1 0 1	1	0 0	1 0 0	0	1 1 0	1	X X X X
CMPLEU R3, R1, R5	0 0 1	1 0 1	1	0 1	1 0 1	0	0 1 1	1	X X X X
ADDI R7, R1, -1	0 0 1	x x x	0	0 0	1 0 0	0	1 1 1	1	F F F F
ANDI R2, R3, 0XFF00	0 1 1	x x x	0	0 0	0 0 0	0	0 1 0	1	F F 0 0
NOT R4, R2	0 1 0	x x x	x	0 0	0 1 1	0	1 0 0	1	X X X X
MOVE R1, R5	1 0 1	x x x	x	1 0	0 0 0	0	0 0 1	1	X X X X
MOVEI R3, 0XFA02	x x x	x x x	0	1 0	0 0 1	0	0 1 1	1	F A 0 2
IN R2	x x x	x x x	x	x x	x x x	1	0 1 0	1	X X X X
OUT R4	1 0 0	x x x	x	x x	x x x	x	x x x	0	X X X X
ANDI -, R3, 0x8000	0 1 1	x x x	0	0 0	0 0 0	x	x x x	0	8 0 0 0
NOP	x x x	x x x	x	x x	x x x	x	x x x	0	X X X X
IN R2 // OUT R4	1 0 0	x x x	x	x x	x x x	1	0 1 0	1	X X X X

Fig. 8.9 Ejemplos de acciones que se pueden hacer en la UP en un ciclo y las palabras de control de 33 bits asociadas a cada acción

en el bus RD-IN, el primero en el ciclo k y el siguiente en el k+2. Esto se puede hacer con una secuencia de tres acciones: en el ciclo k se ejecuta una acción `IN`, al ciclo siguiente la `NOP` y en el k+2 la otra `IN`. No obstante, si es posible adelantar algún trabajo, sería más eficiente hacerlo en el ciclo k+1, en vez de la `NOP`. La palabra de control para la acción `NOP` se muestra en la penúltima fila de la figura 8.9. Por otro lado, también se puede usar para el mismo fin que la `NOP` cualquier acción que no escriba el registro destino, como por ejemplo: `ADD -, R0, R0`. Incluso se puede usar una acción que aun escribiendo el registro destino lo deje con el mismo valor que tenía: lo que es equivalente a no modificarlo. Por ejemplo: `AND R4, R4, R4`, o también `ANDI R7, R7, 0`. No obstante, el mnemotécnico `NOP` es más claro para la semántica de la acción que se desea.

Varias acciones en paralelo. Aunque con muchas restricciones, existen algunas parejas de acciones de tipos distintos que se pueden realizar en paralelo, en el mismo ciclo. Por ejemplo, en un ciclo se puede hacer `IN R2` y `OUT R4` codificando la palabra de control como se puede ver en la última fila de la figura 8.9. Separamos las acciones con el símbolo `//` para indicar que se realizan en paralelo.

Otro ejemplo es hacer en un solo ciclo `SUB R1, R3, R2` (que no está codificada en la tabla anterior) y `OUT R3`. De hecho, en cada ciclo la salida WR-OUT tiene el valor del registro que se lee por el bus A.

Un ejemplo de dos acciones que no pueden hacerse en el mismo ciclo es `ADD R6,R3,R5` e `IN R2`, ya que sólo hay un bus de escritura en el banco de registros y sólo uno de los dos valores, el que sale de la ALU o el que llega por RD-IN, puede atravesar el multiplexor para estar presente en el bus de entrada D del banco de registros y así escribirse al final del ciclo.

No obstante, para facilitar la comprensión, en los ejemplos que vamos a tratar no nos obsesionaremos por aprovechar este paralelismo de la UPG, que por otro lado sólo se puede aplicar en algunos casos muy concretos. Solamente lo usaremos para hacer, cuando se pueda, `IN // OUT` en el mismo ciclo.

8.6 Del código en lenguaje C al grafo de estados de la UC (sin entrada/salida)

8.6.1 Un ejemplo de PPE para calcular el MCD (sin entrada/salida de datos)

Veamos el ejemplo de implementación de un PPE para calcular el máximo común divisor (MCD) de dos números naturales, como ya hicimos en el capítulo anterior, pero ahora usando la UPG como unidad de proceso. En esta sección suponemos que los dos números sobre los que hay que calcular el MCD se encuentran almacenados en los registros `R0` y `R1` de la UPG antes de que empiece el cálculo y que una vez calculado el MCD hay que dejarlo en el registro `R0`. En la siguiente sección ya nos ocuparemos de cómo entrar los operandos y cómo sacar el resultado.

Así que tenemos que implementar el siguiente algoritmo (especificado en alto nivel):

```
while (R0 != R1) {
   if (R0 > R1) R0 = R0 - R1;
   else R1 = R1 - R0;
}
```

La figura 8.10 muestra el fragmento de grafo de la UCE que implementa el algoritmo. Los nodos se han dispuesto en vertical y se han numerado, por si nos tenemos que referir a alguno de ellos. Las señales de salida de la UCE, los 33 bits de la palabra de control que debe generar la UCE en cada nodo, se especifican al lado de cada nodo escritas con mnemotécnicos para que resulte más clara la acción que se realiza en la UPG en cada ciclo. Esto facilita el entendimiento del grafo/algoritmo y es la forma de especificar la funcionalidad de la UCE que usaremos asiduamente a partir de ahora, porque se parece más a la escritura de un programa, que es hacia dónde vamos. Veamos en detalle de esta implementación.

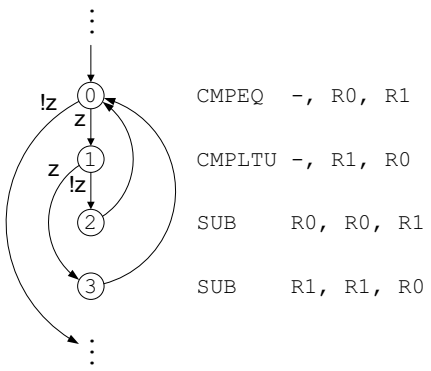


Fig. 8.10 Fragmento del grafo de la UCE para calcular el MCD de R0 y R1.

La sentencia de alto nivel `while (R0!=R1)` se implementa con el nodo 0 del grafo. Como la ALU de la UPG no dispone de la funcionalidad de comparar si dos operandos son distintos hemos usado la comparación por igualdad, que sí existe en la ALU: `CMPEQ`. El resultado de la comparación no se almacena en ningún registro ya que esto no es necesario: es suficiente que la UC decida en ese mismo ciclo en que se realiza la comparación cuál será el estado siguiente en función del bit `z` que genera la ALU y llega a la UC. Por ello, el estado siguiente al 0 será el 1 (entrando en el cuerpo del bucle `while`) si `R0!=R1`, o lo que es lo mismo si el resultado de la comparación (R0 == R1) es falso. Cuando el resultado de una comparación es falso la ALU genera el valor 0 en los 16 bits de su salida y un 1 en su salida `z`. Por ello, el arco que va del nodo 0 al 1 se ha etiquetado con `z`. Sin embargo, si `z` vale

0 quiere decir que el resultado de la comparación (R0 == R1) es verdadero, lo que equivale a decir que la condición del `while(R0!=R1)` es falsa y se debe salir del bucle `while` y pasar al final del fragmento del grafo, por lo que el arco que va del nodo 0 al final del grafo se ha etiquetado con !z.

La primera acción del cuerpo del bucle `while` es la comparación de la sentencia `if (R0>R1)` que se realiza en el nodo 1. Pero como no existe en la ALU la comparación de mayor no se puede implementar directamente `R0>R1` y entonces lo que hacemos es leer la comparación de derecha a izquierda e implementar la comparación equivalente `R1<R0`, mediante la acción `CMPLTU -, R1, R0`. El estado siguiente del nodo 1 será el 2 (donde se implementa `R0=R0-R1`) si el resultado de la comparación es verdadero: la etiqueta del arco que va del nodo 1 al 2 es !z. Si el resultado es falso no se debe realizar la operación del nodo 2 si no la operación del `else R1=R1-R0`, que se implementa en el nodo 3.

Por último, después de ejecutar el nodo 2 o el 3 se ha terminado el cuerpo del bucle `while`, por lo que el estado siguiente de ambos nodos es siempre el nodo 0.

8.7 Entrada/salida síncrona

Para comunicar datos entre dos subsistemas muy relacionados, dentro de un sistema más grande, se suele usar una comunicación (entrada/salida) síncrona. Los dos subsistemas tienen el mismo reloj y el protocolo de comunicación puede hacer referencia a los ciclos de ese reloj común. Esto es lo que hemos hecho en el capítulo 7, por ejemplo, para entrar una secuencia de 4 datos a un PPE desde un subsistema externo, usando la señal Begin para sincronizar la entrada.

A continuación, vamos a implementar los mismos procesadores de propósito específico que vimos en el capítulo 7 con entrada/salida síncrona, pero usando la UPG como unidad de proceso; esto es, vamos a diseñar el grafo de la UC de esos PPE. Para gestionar la entrada/salida de datos/resultado síncrona se usaran las señales Begin y End como únicas de tipo Control-In y Control-Out de nuestro PPE (ver el esquema general de un PPE con la UPG de la figura 8.5). Así, los PPE de esta sección tienen el esquema general de la figura 8.11.

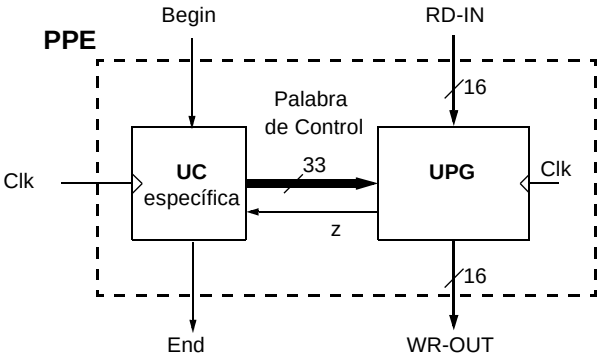


Fig. 8.11 Esquema general de interconexión de la UC con la UPG

Como la UPG ya está diseñada nos centramos en el diseño interno de la UC que especificaremos mediante un grafo de Moore. La UC debe generar en cada ciclo:

- la palabra de control (de 33 bits) y la salida de control End y
- el estado siguiente que es función del estado actual, de la entrada de control Begin y, en general, del bit z que llega a la UC desde la UPG (aunque z no se usa en el primer ejemplo de diseño).

8.7.1 SUMA-4 con la UPG

Vamos a diseñar el procesador de propósito específico, PPE de la figura 8.11, para sumar cuatro números que llegan del exterior del procesador por el bus RD-IN en ciclos consecutivos y sacar el resultado de la suma por el bus de salida WR-OUT. La entrada/salida de datos/resultado se realiza con el siguiente protocolo síncrono usando las señales Begin y End:

- En el ciclo en el que Begin vale 1 en bus de entrada RD-IN se encuentra el primer número a sumar y en los tres ciclos siguientes los otros tres números, a razón de uno por ciclo.
- Una vez realizada la suma, el PPE debe mantener el resultado en el bus de salida WR-OUT durante un ciclo y poner la señal End a 1 durante ese mismo ciclo (indicando que se ha terminado la suma actual, que el resultado está en ese ciclo en el bus WR-OUT y que el PPE puede iniciar en ese mismo ciclo la entrada del primer número de una nueva secuencia de cuatro para ser sumados (si en ese ciclo Begin vale 1).
- El PPE debe ignorar la señal Begin desde después del ciclo en que Begin vale 1 hasta el ciclo anterior en el que End vale 1. Esto es, no se abortará la entrada de datos ni el cálculo de la suma en curso para empezar a sumar otros cuatro números.
- Si Begin vale 1 durante el mismo ciclo en el que End vale 1 (indicando que ya termino la suma de los cuatro números) se deberá iniciar la suma de otros cuatro números (el primero de los cuales está en RD-IN en este mismo ciclo).

En la implementación de propósito específico del capítulo 7, el primer dato se guardaba en el registro (el único necesario en la implementación específica), el segundo, que llegaba justo al ciclo siguiente de llegar el primero, se sumaba con el primero en el mismo ciclo que llegaba y al final del ciclo se almacenaba el resultado en el registro. Con el tercer y cuarto dato se procedía igual que con el segundo. Al ciclo siguiente de llegar el cuarto dato, el resultado se encontraba en la salida: en total se requerían 5 ciclos.

Esto no se puede hacer tan rápido con la actual UPG, ya que (ver figura 8.6):

- no se puede conectar directamente la entrada RD-IN con la ALU sin antes pasar por un registro del banco de registros (lo que lo retrasa al menos un ciclo),
- ni se puede guardar en el banco de registros el dato que llega por la entrada RD-IN en un ciclo y el resultado producido por la ALU en ese mismo ciclo. Esto hace que ni siquiera se pueda estar entrando un dato en el mismo ciclo que se están sumando dos datos que entraron al banco de registros en ciclos anteriores.

Por ello, lo que haremos será entrar primero los 4 datos que llegan uno detrás de otro en ciclos consecutivos (mediante cuatro acciones del tipo `IN Rk`), escribiéndolos al final de cada ciclo en un registro `Rk` diferente, y después realizar las 3 sumas necesarias (mediante tres acciones del tipo `ADD Rd, Ra, Rb` con los registros adecuados).

La figura 8.12 muestra el grafo de estados de la UC para resolver este problema. Cómo el valor de la entrada `z` es indiferente para calcular el estado siguiente para cualquiera de los nodos del grafo, está señal no aparece en los arcos del grafo. Las salidas de la unidad de control para cada estado se han denotado a la derecha de cada nodo: la palabra de control con mnemotécnicos y separado por el símbolo // el valor de la señal `End`.

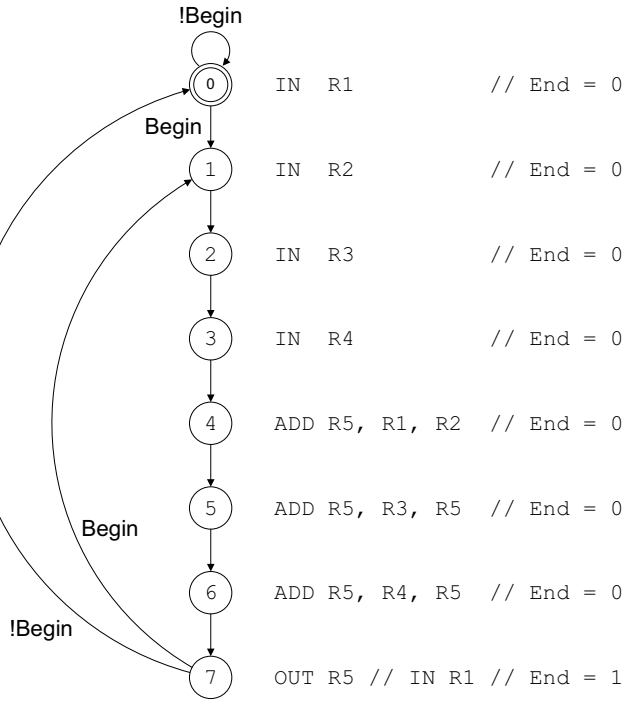


Fig. 8.12 Grafo de estados de la UC del PPE Suma-4 con entrada/salida síncrona.

A modo de ejercicio, traducid cada una de las palabras de control especificada con mnemotécnicos a su vector de 33 bits (ver solución en figura 8.13).

Como ya se anunció en un párrafo anterior, el tiempo de sumar los cuatro números usando la UPG es mayor que en el diseño totalmente específico del capítulo anterior: 8 ciclos frente a 5. Esto es así en casi todos los ejemplos, la implementación con una UP más específica puede ser más rápida que con la UPG. Por contra, la UPG puede servir para resolver diferentes problemas con sólo cambiar la UC.

	Palabra de Control de 33 bits								
Mnemotécnicos	@A	@B	Rb/N	OP	F	In/Alu	@D	WrD	N (hexa)
IN R1	x x x	x x x	x	x x	x x x	1	0 0 1	1	X X X X
IN R2	x x x	x x x	x	x x	x x x	1	0 1 0	1	X X X X
IN R3	x x x	x x x	x	x x	x x x	1	0 1 1	1	X X X X
IN R4	x x x	x x x	x	x x	x x x	1	1 0 0	1	X X X X
ADD R5, R1, R2	0 0 1	0 1 0	1	0 0	1 0 0	0	1 0 1	1	X X X X
ADD R5, R3, R5	0 1 1	1 0 1	1	0 0	1 0 0	0	1 0 1	1	X X X X
ADD R5, R4, R5	1 0 0	1 0 1	1	0 0	1 0 0	0	1 0 1	1	X X X X
OUT R5 // IN R1	1 0 1	x x x	x	x x	x x x	1	0 0 1	1	X X X X

Fig. 8.13 Tabla de traducción de mnemotécnicos a bits de la palabra de control de la UC del PPE Suma-4

8.7.2 SUMA-n con la UPG

Diseñamos a continuación el grafo de la UC que junto a la UPG implemente el procesador de propósito específico (ver figura 8.11) con la misma funcionalidad que el denominado SUMA-n en el del capítulo 7. El enunciado del capítulo anterior decía así: diseñad un procesador que obtenga la suma de *n* números naturales codificados en binario con 16 bits que llegan al procesador ciclo a ciclo por la entrada RD-IN. En el primer ciclo, cuando la entrada de control Begin vale 1, en la entrada de datos se encuentra el valor *n*. En los *n* siguientes ciclos están disponibles, en esa misma entrada, los valores de los *n* números, a razón de uno por ciclo. La suma debe estar disponible en la salida WR-OUT cuando finalice el cálculo, lo que se indicará poniendo la salida de control End a 1 durante ese mismo ciclo. Como en el ejemplo anterior no se aborta la suma de n números en curso y no se ignora la señal Begin en el ciclo en el que End vale 1.

En la implementación del PPE en el capítulo anterior teníamos 2 unidades aritméticas trabajando en paralelo. En un mismo ciclo el sumador sumaba el nuevo número que llegaba por la entrada RD-IN con el contenido del registro acumulador, mientras que el decrementador restaba 1 al contador para saber los datos que faltaban por llegar (y por sumar).

Esto no puede hacerse a esta velocidad usando la UPG, ya que sólo tiene una unidad funcional (la ALU) que deberá ser usada tanto para sumar como para decrementar, pero no en el mismo ciclo las dos cosas. En un ciclo la ALU decrementará y en otro sumará. Además, como se vio en el ejemplo anterior, no se puede almacenar en el banco de registros en el mismo ciclo el resultado de la ALU y el dato presente en RD-IN. Con todo ello, resulta imposible implementar el sistema con la UPG si los datos llegan a la velocidad de uno por ciclo. Necesitamos varios ciclos entre la llegada de un dato y la del siguiente, un ciclo para entrar el dato, otro para sumar y otro para decrementar y saber si faltan más datos). Esto nos obliga a modificar el enunciado original en cuanto a la sincronización de la llegada de

los datos. En la nueva especificación, los datos han de llegar según nos convenga en nuestro diseño y por lo tanto habría que modificar el diseño del sistema que los proporciona.

Por otro lado, no almacenaremos primero los datos en el banco de registros y luego los sumaremos, como se hizo en el ejemplo SUMA-4, ya que esto supondría limitar *n* al tamaño del banco de registros.

La figura 8.14 muestra el grafo de la UC que junto a la UPG resuelve este problema.

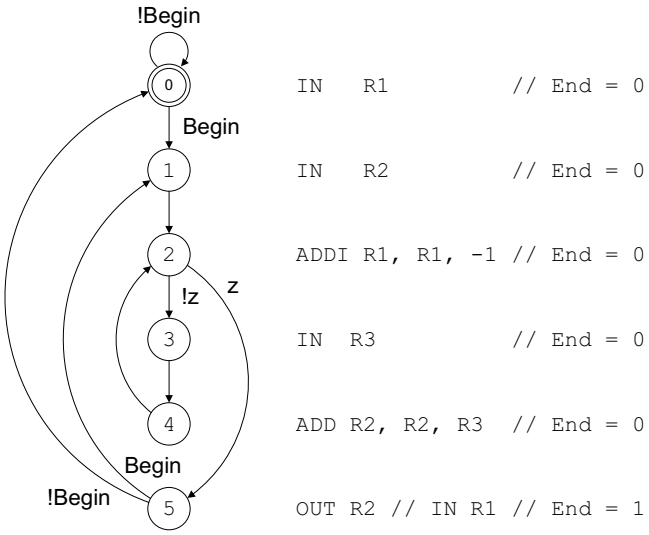


Fig. 8.14 Grafo de estados de la UC del PPE SUMA-n con entrada/salida síncrona

A la vista del grafo que hemos obtenido, la llegada síncrona del valor *n* y de los *n* operandos para que el sistema funcione correctamente debe formularse como sigue.

La señal End se pondrá a 1 durante un ciclo indicando al PPE que debe comenzar y que los datos se encuentran en el bus RD-IN de entrada con la siguiente secuencia:

- Durante el mismo ciclo que Begin vale 1 se encuentra el valor *n*. Supongamos que este es el ciclo *c*.
- Al ciclo siguiente, durante el ciclo *c*+1, se encuentra el primer dato a sumar.
- En el ciclo *c*+3 se encuentra el segundo dato (si *n*>1), y a partir de este momento cada 3 ciclos se encuentra un nuevo dato (ciclos *c*+6, *c*+9, *c*+12...).

Ejercicio:

Efectuar las modificaciones oportunas en el grafo de la figura 8.14 para que el procesador de la sección anterior funcione correctamente con la siguiente sincronización de los datos que llegan por RD-IN.

- Durante el mismo ciclo que Begin vale 1 en RD-IN se encuentra el valor *n*. Supongamos que este es el ciclo *c*.
- A partir del ciclo *c*, cada 3 ciclos llega un dato a sumar. Esto es, el primer dato a sumar llega en el ciclo *c*+3, el segundo en el *c*+6, el tercero en el *c*+9,...

8.7.3 MCD con la UPG

El PPE para calcular el MCD que diseñamos en el capítulo 7 tenía dos buses de entrada de datos y en un mismo ciclo, en el que la señal Begin vale 1, se encontraban los dos operandos uno en cada bus. Ahora, como en la UPG sólo hay un bus de entrada, RD-IN, los dos operandos requerirán dos ciclos para entrar a la UPG. Vamos a cambiar el protocolo de la entrada y salida de datos, respecto al usado en los dos ejemplos anteriores y en el del MCD del capítulo anterior. Ahora el protocolo es el siguiente:

- El ciclo en el que Begin vale 1 indica que al ciclo siguiente en el bus RD-IN se encuentra el primer operando y al siguiente de este el segundo.
- Un ciclo antes de que el MCD se encuentre en el bus de salida WR-OUT la señal End deberá valer 1 durante un ciclo.
- La señal Begin se ignorará durante los ciclos en que entran los dos operandos y se realiza el cálculo, volviéndose a considerar durante el ciclo en el que End vale 1 y siguientes, hasta que entren los nuevos datos.

El grafo de la figura 8.15 muestra los tres primeros nodos del algoritmo, encargados de la sincronización de inicio y de la entrada de los dos operandos. A diferencia de los ejemplos anteriores, durante el nodo 0 no hace falta ir cargando el primer operando ya que aunque Begin valga 1 durante este ciclo el primer operando no se encuentra en RD-IN durante este ciclo, sino al siguiente. Por ello, en el nodo 0 no se hace nada en la UPG (acción NOP).

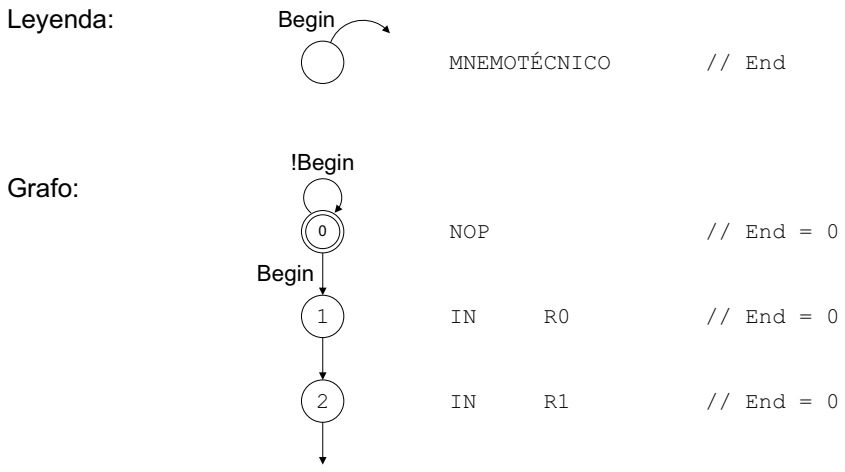


Fig. 8.15 Tres primeros nodos del grafo que calcula el MCD con entrada/salida síncrona y operandos y resultado retrasados un ciclo respecto de Begin y End.

Después de estos tres nodos vendría el fragmento de grafo de la figura 8.10 y finalmente la sincronización de finalización y salida del resultado: véase el grafo completo en la figura 8.16. Se observa que cuando se sabe que termina el cálculo del MCD (nodo 3, en el ciclo en el que z vale 0) hay que ir a un nodo en el que End valga 1, que no puede ser ni el 3 ni el 4 ni el 5 pues en otras ocasiones se está en estos ciclos sin haber terminado el cálculo, y por la misma razón tampoco puede ser ni el 0, 1 o 2. Hace falta poner un nodo para ello, que es el nodo 7, en el que se está un ciclo en el que no puede hacerse nada útil (el `OUT` tiene que hacerse al ciclo siguiente), por lo que la acción es `NOP`. No obstante, según el protocolo de sincronización del enunciado, en este nodo en el que se pone End a 1 ya hay que observar Begin y actuar en consecuencia: que el estado siguiente sea el nodo 0 (si Begin vale 0) o el nodo 1 (si Begin vale 1). Como al ciclo siguiente de estar en el nodo 7 hay que hacer la acción `OUT R0` para mostrar el resultado en el bus WR-OUT, pondremos esta acción tanto en el nodo 0 (en vez de la `NOP` original) como en el nodo 1 (en paralelo con la acción `IN R0`). Estas acciones `OUT` no molestan cuando estamos en el nodo 0 y 1 la primera vez, cuando todavía no se ha realizado ningún calculo previo del MCD. Tampoco molesta que después de terminar un cálculo en WR-OUT esté el resultado durante más ciclos que el estrictamente necesario (que es el ciclo siguiente de que End valga 1).

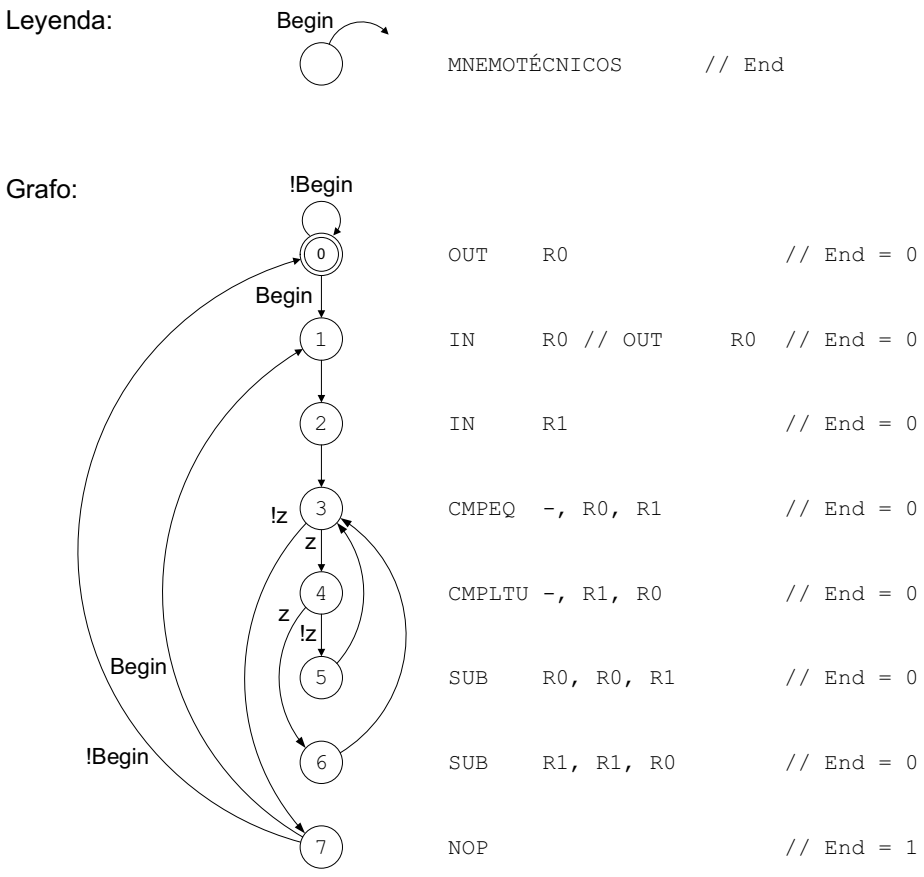


Fig. 8.16 Grafo completo que calcula el MCD con entrada/salida síncrona y operandos y resultado retrasados un ciclo respecto de Begin y End.

Comentario final

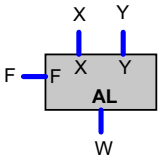
Es importante resaltar que en el proceso que nos lleva, paso a paso, de un diseño específico para resolver cada problema a un único procesador de propósito general, que nos servirá para resolver cualquier problema, ganamos en generalidad y en tiempo de diseño a costa de perder en tiempo de ejecución, en tiempo de resolución de cada problema concreto. Con la UPG, la resolución de un problema requerirá, en general, más ciclos que con una unidad de proceso de propósito específico, como las diseñadas en el capítulo anterior, en las que puede haber mucho hardware disponible para ser usado en paralelo, en un mismo ciclo. También es muy probable que el tiempo de ciclo del procesador de propósito general sea mayor que el que podríamos obtener con una implementación de propósito específico para un problema.

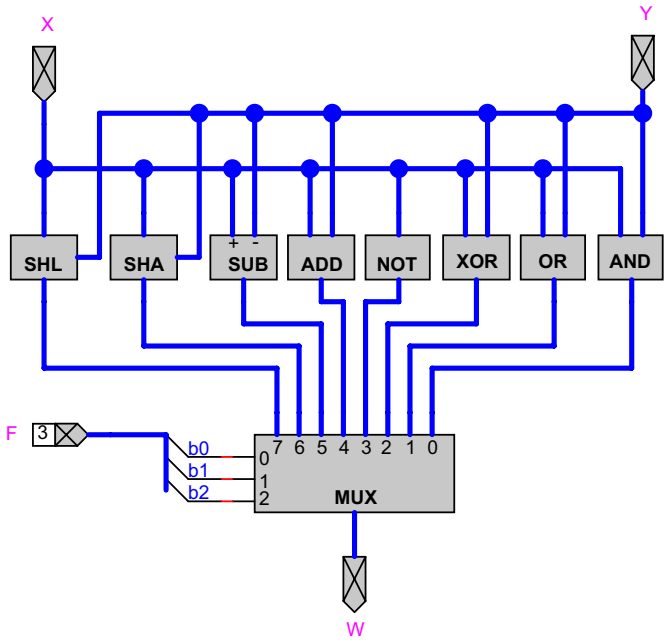
ANEXO: Fichas de descripción de los dispositivos que forman la UPG

16 Bit-wide Arithmetic and Logic Operators — AL

Description:
The 16-bit output W is the arithmetic or logic function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y.
The functions are:

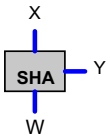
F	W
000	AND (X, Y)
001	OR (X, Y)
010	XOR(X, Y)
011	NOT (X)
100	ADD (X, Y)
101	SUB (X, Y)
110	SHA (X, Y)
111	SHL (X, Y)





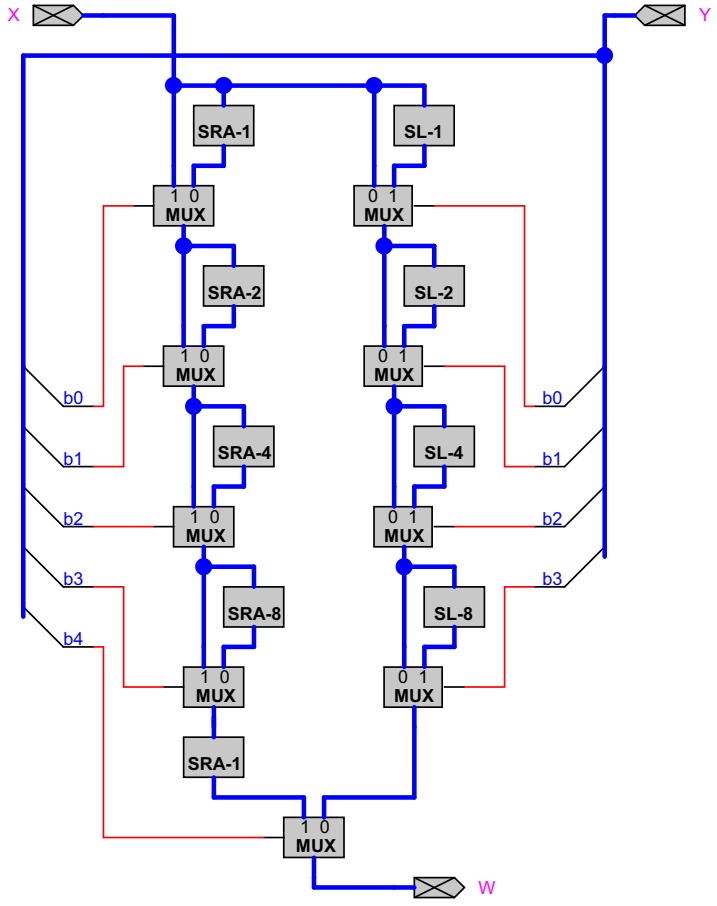
16 Bit-wide Arithmetic Shifter — SHA

Description:
The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b4:b0) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHA shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHA shifts to the right bit X(15) is sign-extended into the most significant bits of W and the less significant bits of X are lost.



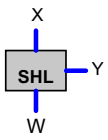
Arithmetic Operation:
Interpreting X, Y(b4:b0) and W as signed integers,
if ((-2**15) <= (Xs * (2**Y(b4:b0)s)) <= (2**15)-1) then Ws = Xs * (2**Y(b4:b0)s)
(The integer división of powers of 2 (when Y(b4:b0)s is negative) is done whith positive remainder)

Bit-level Logical Operation:
if (Y(b4:b0)s >= 0) then
W(bi) = 0 for i = 0 to Y(b4:b0)s -1
W(bi) = X(bi-Y(b4:b0)s) for i = Y(b4:b0)s to 15
if (Y(b4:b0)s < 0) then
W(bi) = X(bi-Y(b4:b0)s) for i = 0 to -Y(b4:b0)s - 1
W(bi) = X(b15) for i = -Y(b4:b0)s to 15



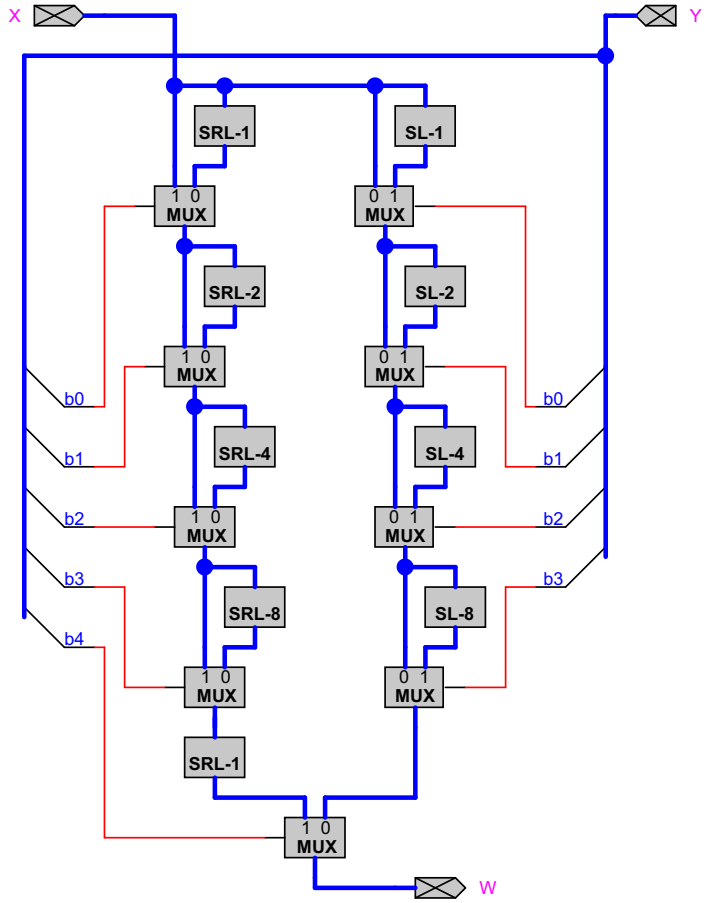
16 Bit-wide Logic Shifter — SHL

Description:
The 16-bit output W is the result of shifting the 16-bit input X the number of bits coded by bits 0 to 4 of the 16-bit input Y. Y(b0:b4) represents a signed integer in two´s-complement.The number of shifting bits is the absolute value of the number represented in Y. If this value is positive, the shifting is to the left and if it is negative the shifting is to the right. When SHL shifts to the left the less significant bits of W are set to 0 and the most significant bits of X are lost. When SHL shifts to the right the most significant bits of W are set to 0 and the less significant bits of X are lost.

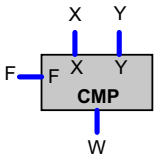


Arithmetic Operation:
Interpreting X and W as unsigned integers and Y as signed integer
if ((Xu * (2**Y(b0:b4)s)) <= (2**16)-1)) then Wu = Xu * (2**Y(b0:b4)s)

Bit-level Logical Operation:
if (Y(b0:b4)s >= 0) then
W(bi) = 0 for i = 0 to Y(b0:b4)s -1
W(bi) = X(bi-Y(b0:b4)s) for i = Y(b0:b4)s to 15
if (Y(b0:b4)s < 0) then
W(bi) = X(bi-Y(b0:b4)s) for i = 0 to -Y(b0:b4)s - 1
W(bi) = 0 for i = -Y(b0:b4)s to 15



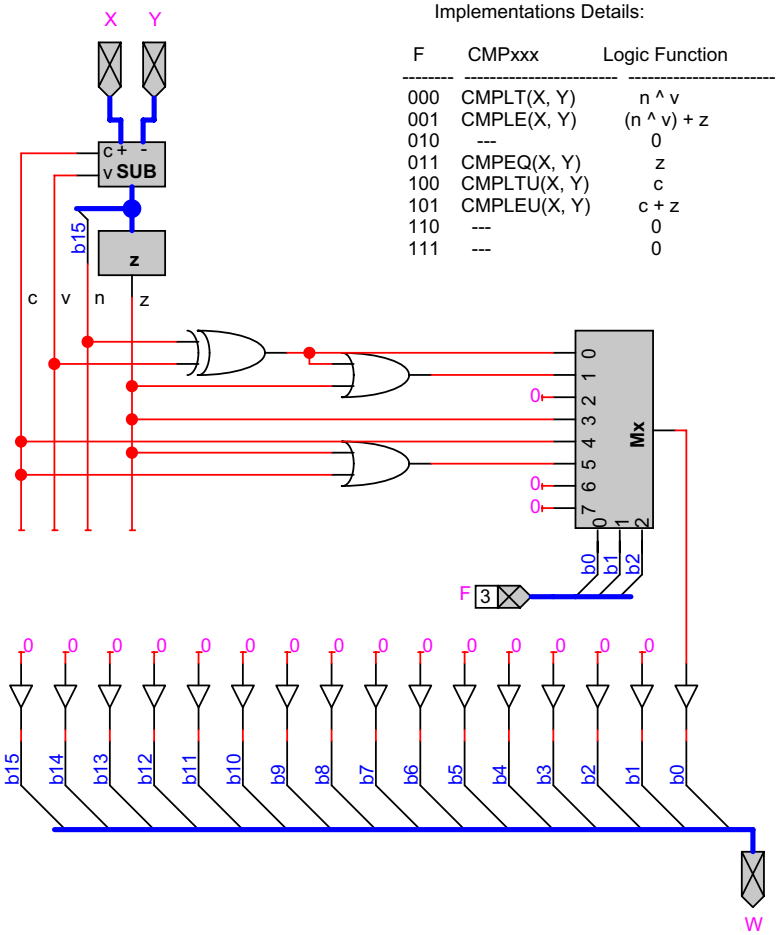
16 Bit-wide Signed and Unsigned Comparator — CMP



Description:

The 16-bit output W has only two posible values as the result of a comparison: true or false. True is coded as W(b0) = 1 and false as W(b0) = 0.
For all the cases W(bi) = 0 for i = 1 to 15. The type of comparison and the consideration of the 16-bit inputs X and Y as signed or as unsigned integers is chosen by the 3-bit selection input F according to the next table:

F	W	CMPxx(X, Y)	Name
000	Xs < Ys	CMPLT(X, Y)	Less Than (Signed)
001	Xs <= Ys	CMPLE(X, Y)	Less than or Equal (Signed)
010	---	---	
011	X ==Y	CMPEQ(X, Y)	Equal
100	Xu < Yu	CMPLTU(X, Y)	Less Than Unsigned
101	Xu <= Yu	CMPLEU(X, Y)	Less than or Equal Unsigned
110	---	---	
111	---	---	



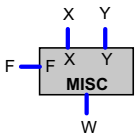
Implementations Details:

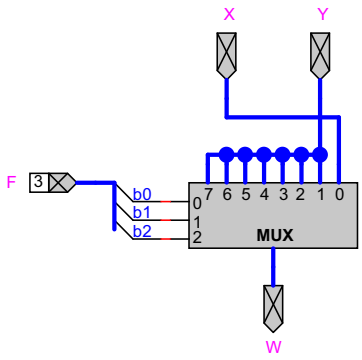
F	CMPxxx	Logic Function
000	CMPLT(X, Y)	n ^ v
001	CMPLE(X, Y)	(n ^ v) + z
010	---	0
011	CMPEQ(X, Y)	z
100	CMPLTU(X, Y)	c
101	CMPLEU(X, Y)	c + z
110	---	0
111	---	0

16 Bit-wide Miscellaneous Operators (Chapter 8) **MISC**

Description:
The 16-bit output W is the function chosen by the 3-bit selection input F, applied to the 16-bit inputs X and Y. The functions are:

F	W
000	X
001	Y
010	---
011	---
100	---
101	---
110	---
111	---





Zero Detector — Zero

Description:
The binary output z takes the value 1 if all the bits of the 16-bit input X are 0, otherwise z takes the value 0.

Bit-level Logical Operation:
if (X(bi) = 0 for i = 0 to 15) then z = 1 else z = 0