7. Procesadores de Propósito Específico

Juan J. Navarro

Primera versión: 07-11-2000
Última actualización: 8-10-2011

7.1 Introducción

En este tema tratamos circuitos lógicos secuenciales de más complejidad que los estudiados hasta ahora, circuitos que procesan palabras de n bits (usaremos n = 8 o n = 16 en los ejemplos). A estos circuitos los denominamos procesadores de propósito específico (PPE). Estos circuitos secuenciales requieren muchos estados y/o transiciones entre estados, de forma que resultaría inviable aplicar las técnica de análisis y síntesis vistas hasta ahora. Es más, la simple especificación de todo un PPE mediante un grafo carece de sentido.

A lo largo de la explicación utilizaremos el siguiente ejercicio a modo de ejemplo: *diseñar un circuito lógico secuencial que realice la suma módulo 2^8 de una secuencia de 4 números naturales codificados en binario con 8 bits cada uno* (Figura 1(a)). Los números a sumar llegan al sistema por la entrada *DATO* (de 8 bits) a razón de un número por ciclo, comenzando por el ciclo en el que la señal de entrada *Ini* vale 1. Una vez terminado el cálculo, el resultado estará disponible en la salida *RESULT (de 8 bits)* durante un ciclo, en el cual el circuito pondrá la salida *Fin* a 1.

La complejidad de un sistema como éste crece exponencialmente con n (siendo n el número de bits con el que se codifican los números, n=8 en el ejemplo). El grafo de Moore de este sistema requeriría $1 + 4 * 2^n$ estados (1 estado inicial de espera y 2^n estados para cada uno de los resultados parciales de las 4 sumas).

La visión que inicialmente tenemos del circuito a diseñar es la de una caja negra con unas entradas y salidas de información (además de la entrada de reloj *CLK*) -ver Figura 1(b). El funcionamiento del circuito se describe indicando qué información entra y cuándo entra por las señales de entrada, cómo se ha de transformar (operar) esa información y finalmente qué resultados calcula y cuándo saldrán éstos por las señales de salida.

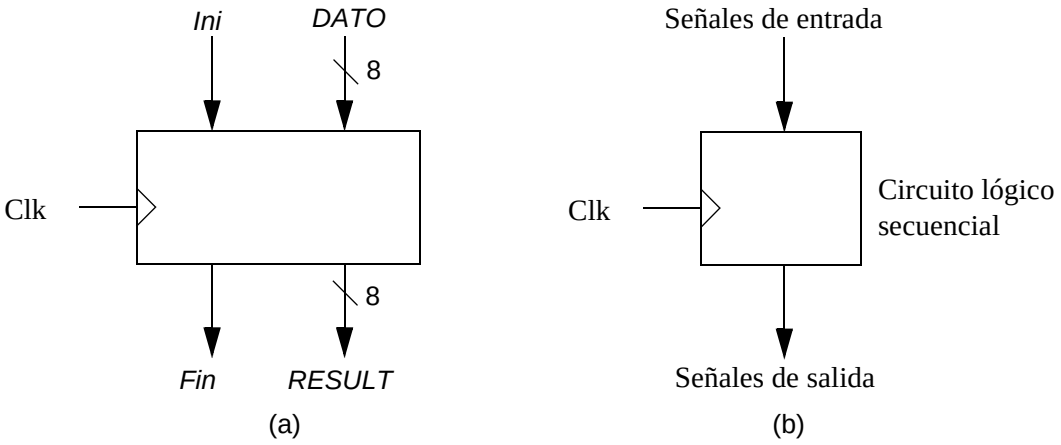


Figura 1. Procesador de propósito específico.

Para refinar más esta descripción del circuito, podemos diferenciar dos tipos de entradas/salidas: de datos y de control. En nuestro ejemplo, *Ini* y *Fin* tienen una funcionalidad típica de algunas entradas/

salidas de control: la validación de los datos de entrada/salida. Cuando la entrada Ini vale 1 le indica al circuito que el número presente durante ese ciclo en la entrada *DATO* es el primero de la secuencia de números que debe sumar. El circuito informa que ya ha terminado el cálculo y que el resultado está disponible en la salida de datos *RESULT*, poniendo un 1 en la salida de control *Fin*.

A diferencia de la información de control, que suele requerir pocos bits, las entradas/salidas de datos codifican información con bastantes bits (8, 16, 32...) para representar, por ejemplo, un número entero codificado en complemento a 2.

Desde el punto de vista de entradas/salidas, podemos especificar el funcionamiento del circuito usando un lenguaje más formal que el textual empleado hasta ahora. A continuación se especifica nuestro ejemplo con uno de estos lenguajes. Es importante ver la diferencia que se hace entre las entradas/ salidas y la información presente en ellas en determinados ciclos de reloj. Así, *Ini* es una entrada de control e *Ini*(*c*) indica el valor presente en *Ini* en el ciclo *c*. Las operaciones a realizar por el circuito se especifican mediante algoritmos o expresiones algebraicas que operan con variables. Las variables de entrada toman su valor de las entradas de datos en los ciclos que indican las entradas de control.

MÓDULO "SUMADOR DE 4 NÚMEROS"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	8 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = \left(\sum_{i=0}^{3} DATO(c1+i) \right) mod(2^8)$$

$$Fin(c2) = 1$$

siendo *DATO(c1 + i)*, para *0* <= *i* <= *3*, un número natural con rango [0, 255] codificado en binario con 8 bits.

FIN MÓDULO

En nuestro ejemplo, la salida está disponible en el ciclo genérico c2, que no se ha relacionado con el ciclo c1. Esto es así porque en la especificación del circuito no se quiere fijar cuántos ciclos ha de tardar el cálculo (se deja esta decisión al diseñador). Si se quisiera especificar qué, por ejemplo, el resultado debe estar disponible 4 ciclos después del ciclo en el que *Ini* vale 1, se hubiera indicado como sigue:

RESULT(c1 + 4) = ...
Fin(c1 + 4) = 1

7.2 Diseño de Procesadores de Propósito Específico con Unidad de Proceso (UP) y Unidd de Control (UC)

El trabajo del diseñador comienza a partir de una descripción más o menos formal del circuito desde el punto de vista de entradas/salidas. Además, el diseñador debe tener en cuenta ciertas restricciones (como pueden ser el número de ciclos que tarda el circuito en entregar el resultado, el tiempo de ciclo o el número y tipo de circuitos a utilizar -puertas, bloques).

A efectos de diseño, un sistema lógico secuencial complejo como es un procesador de propósito específico, que procesa palabras de n bits, se descompone en dos subsistemas (ambos secuenciales): *Unidad de Proceso* (UP) o camino de datos -data path- y *Unidad de Control* (UC). La UP almacena y transforma (opera) los datos hasta obtener los resultados y la UC controla las operaciones que se realizan en la UP y su secuenciamiento correcto.

Las entradas/salidas de datos están conectadas a la UP y las de control a la UC. La UC y la UP se comunican entre si mediante la "*Palabra de Control*" y la "*Palabra de Condición*". En la Figura 2 se muestran los subsistemas de control y de proceso y su interconexión. La UC genera en cada ciclo una Palabra de Control que gobierna las operaciones que han de realizarse en la UP durante ese ciclo. A su vez, la UP genera en cada ciclo una Palabra de Condición que informa a la UC de la ocurrencia, en la UP, de ciertos eventos que son importantes para el secuenciamiento de las operaciones. Por ejemplo, la UP informará a la UC si el resultado de una operación es cero o distinto de cero cuando el tipo de operación a realizar en el ciclo siguiente dependa de ese resultado.

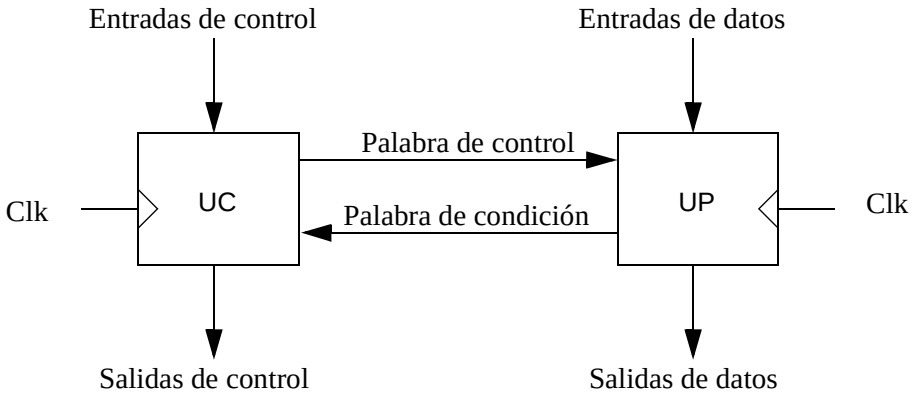


Figura 2. Conexión entre unidad de control y de proceso.

7.2.1 Unidad de Proceso

La UP es un circuito secuencial que diseñaremos a nivel de bloques mediante un diseño *ad-hoc*. Es decir, no usaremos el método sistemático visto en el capítulo anterior, que comienza construyendo un grafo de estados a partir de la descripción del circuito. Diseñaremos una UP formada por elementos de almacenamiento (registros, bancos de registros, memorias) interconectados entre sí a través de bloques combinacionales (sumadores, restadores, desplazadores aritméticos, incrementadores, comparadores, ALUs, codificadores, decodificadores, multiplexores...) y puertas lógicas.

En la Figura 3 se muestra una posible UP para nuestro ejemplo. Un sumador es suficiente para sumar los 4 números, ya que estos llegan uno por ciclo y podemos reutilizar el sumador. Dicho de otra forma, poniendo más de un sumador no conseguiríamos reducir el tiempo de realizar las cuatro sumas. En el primer ciclo guardaremos en el registro **REG** el primer número. Al ciclo siguiente sumaremos el contenido de **REG** (el primer número) con el segundo número, que durante este ciclo está presente en la entrada **DATO**. Al final del ciclo el resultado de esta suma se guardará en **REG**. Así se ira acumulando al resultado parcial un número en cada ciclo. Al final del cuarto ciclo se cargará en **REG** el resultado de sumar los cuatro números.

7.2.2 Unidad de Control

La UC también es un circuito secuencial. Su complejidad, en cuanto al número de bits de las entradas/ salidas y el número de estados, es más reducida que la de la UP. Por ello la UC se especificará mediante un grafo de estados según el modelo de Moore. Las salidas de un autómata de Moore, en

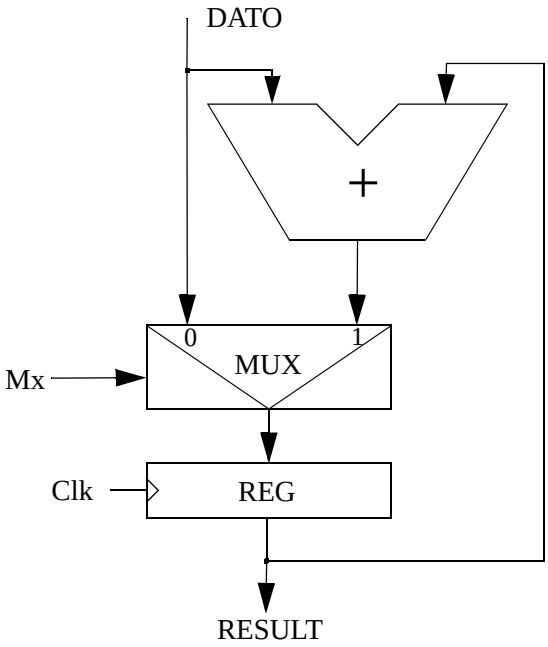


Figura 3. Unidad de proceso para la suma de cuatro números.

nuestro caso la *Palabra de Control* -ver Figura 2- que gobierna a la UP y las salidas de control del sistema son en cada ciclo función únicamente del estado actual. El estado siguiente de la UC es función de las entradas de control del sistema, de la *Palabra de Condición* que le llega de la UP y de su estado interno.

En todos los ejercicios de este capítulo será suficiente con especificar el grafo de estados de la UC. Posibles implementaciones de autómatas de Moore ya se han estudiado en el capítulo anterior.

La unidad de control de nuestro ejemplo solamente tiene un bit de entrada (*Ini*), ya que no necesita saber ninguna condición de la UP para generar el secuenciamiento adecuado de las palabras de control. Las salidas de la UC son: el bit *Fin* (salida de control del sistema) y el bit *Mx* (*palabra de control*). La Figura 4 muestra un cronograma del funcionamiento del circuito (los números a sumar son 23 + 5 + 12 + 18). En la Figura 5 puede verse el grafo de estados de la UC.

Algunos comentarios sobre el grafo de estados son los siguientes:

1. El registro REG de la UP se carga todos los ciclos con lo que hay a su entrada, ya que no tiene señal de Load. Esto, que es perfectamente válido para este ejemplo, no es general. En muchos otros casos, un valor almacenado en un registro debe permanecer en él durante varios ciclos. Para que esto ocurra es necesario que la UC genere una señal de *Load* que indica en qué ciclos hay que cargar el registro y en cuáles no.
2. Al final de cada ciclo en que la UC está en el estado de *Espera* se carga el valor de la entrada *DATO* en REG (ya que en el estado de espera, *Mx* = 0). Este valor no se usa para nada si *Ini* vale 0, pero si *Ini* vale 1 la UC pasará en el ciclo siguiente a otro estado en el que se sumará el *dato 2* con el *dato 1* que se almacenó en REG al final del ciclo anterior (cuando el autómata estaba en el estado de espera).

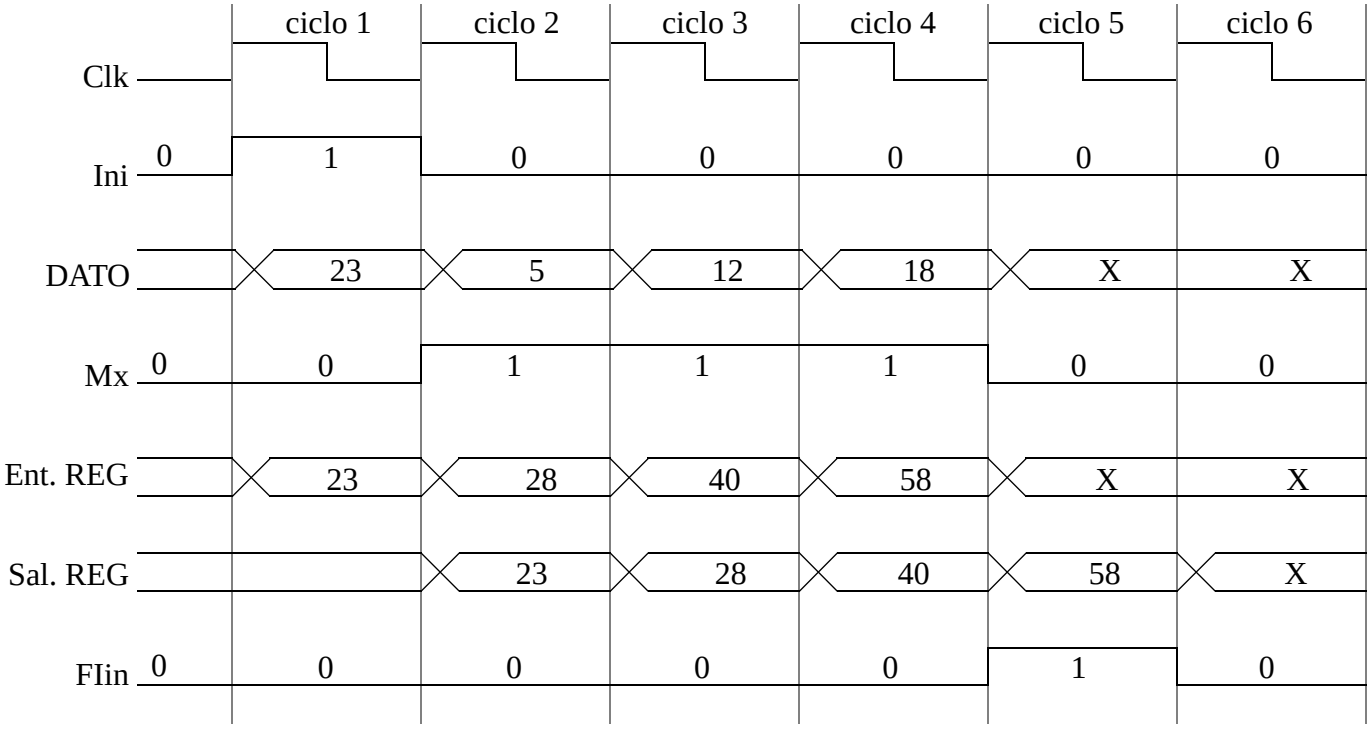


Figura 4. Cronograma del sistema para la suma de cuatro números.

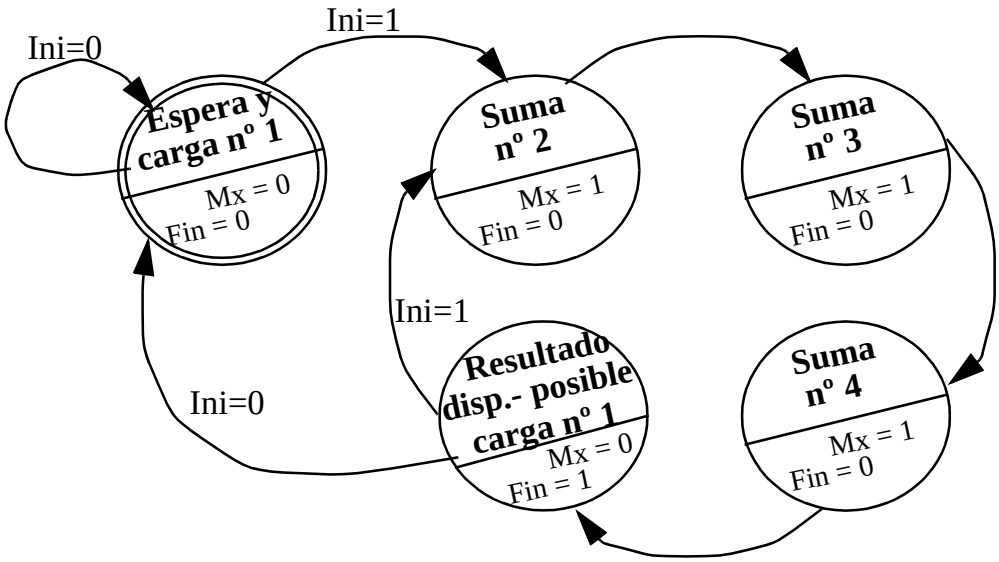


Figura 5. Diagrama de Moore para el sumador de cuatro números.

7.3 Tiempos de ciclo y de ejecución

Veamos algunas consideraciones importantes acerca de la evolución temporal de las señales en el circuito y su repercusión en el tiempo de ciclo.

Supondremos que el circuito a diseñar está inmerso en un sistema más grande. Esto quiere decir que las señales de entrada de nuestro circuito son las salidas de otros circuitos, sincronizadas con el mismo reloj. Asimismo, las salidas de nuestro circuito servirán de entradas a otros circuitos también sincronizados con el mismo reloj.

Tanto la UP como la UC son circuitos secuenciales y como tales están formados por biestables (registros) conectados entre sí a través de circuitos combinacionales (puertas, sumadores, multiplexores, memoria ROM, etc.). Los valores que hay en las entradas de los registros y biestables son capturados por éstos cuando llega el flanco ascendente de reloj.

El tiempo de ciclo mínimo para que funcione correctamente el PPE se calcula como el caso de un circuito secuencial: se encuentra el camíno crítico y el tiempo de ciclo mínimo es el tiempo del camino crítico. La única cosa diferente es que en este circuito secuencial el diseño está partido en dos grandes bloques: UC y UP. El cámino crítico puede ir de un biestable de la UC a un registro de la UP, por ejemplo. También puede ir de un biestable de la UC a la UP, pasar por algunos bloques combinacionales de la UP (por ejemplo un multiplexor, un comparador) y volver a la UC (a traves de una señal de la palabra de condición) pasar por la ROM de la UC que calcula el estado siguiente y terminar en un biestable de la UC. También puede ir de un registro de la UP a un registro de la UP o a un biestale de la UC. El camino crítico tambien puede ir de una entrada a un biestable o registro o de un biestable o registro a una salida. Enfín, es como el caso de un secuencial, no tiene nada de particular.

El tiempo de ciclo es muy importante, ya que nos permitirá comparar diferentes diseños desde el punto de vista del *tiempo de ejecución* de la operación que debemos diseñar. El *tiempo de ejecución* -**Te**- (en nuestro ejemplo el tiempo necesario para sumar los cuatro números) es igual al número de ciclos -**Nc**- que se tarda en realizar el cálculo multiplicado por el tiempo de ciclo ***Tc***:

Te = ***Nc*** * ***Tc***.

Existen diferentes posibles diseños de un circuito, todos con la misma funcionalidad pero no necesariamente con el mismo **Te**. Debemos elegir el mejor según un criterio especificado, como por ejemplo minimizar **Te** teniendo en cuenta ciertas restricciones en el uso del hardware.

ANEXO A: Ejemplo, el cálculo del Máximo Común Divisor

El objetivo de este anexo consiste en resolver un ejemplo de diseño más complejo que el de la suma de los 4 números. El ejemplo consiste en diseñar, utilizando el algoritmo de Euclides, un circuito capaz de calcular el Máximo Común Divisor de dos números enteros X e Y codificados en complemento a 2 con 16 bits cada uno:

MÓDULO "MCD"

ENTRADAS/SALIDAS

Entradas	de datos:	X, Y	16 bits
	de control:	Ini	1 bit
Salidas	de datos:	MCD	16 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = MCD(X, Y)$$

$$Fin(c2) = 1$$

donde *X(c1), Y(c1) y MCD(c2)* son números enteros con 16 bits codificados en complemento a 2.

FIN MÓDULO

El algoritmo de Euclides se basa en la realización sistemática de restas entre los números X e Y. Cuando ambos valores son iguales, el resultado obtenido es el Máximo Común Divisor.

```
Algoritmo de Euclides para MCD(X,Y)
    A:= X; B:= Y;
    mientras (A <> B) hacer
                    si (A > B) entonces     A:= A-B;
                    sino                    B:= B-A;
    fmientras
    MCD:= A;
```

Unidad de Proceso.

Para implementar este sistema, inicialmente analizaremos qué componentes son necesarios en la Unidad de Proceso (ver Figura 7).

- A y B son dos variables en las que inicialmente se guarda una copia de **X** e **Y**, para posteriormente ser modificadas por las sucesivas restas. Por ello es preciso disponer de dos registros para almacenarlas.
- Para realizar las restas utilizaremos un único restador. No es necesario un segundo restador ya que las restas A-B y B-A se realizan de manera excluyente.
- También será necesario usar un comparador de números naturales en complemento a 2 para poder calcular si A<>B, o si A>B.
- Finalmente serán necesarios varios multiplexores para encaminar los datos.

En esta Unidad de Proceso será necesario crear un camino de datos desde las entradas **X** e **Y** a los registros que almacenan las variables **A** y **B**. Como en estos registros es necesario guardar el resultado de la resta, es imprescindible utilizar un multiplexor. De esta forma podremos seleccionar entre guardar el dato inicial o el resultado de la resta en las sucesivas iteraciones.

Por otra parte, es preciso utilizar dos multiplexores adicionales de forma que con un solo restador se puedan implementar las dos restas (A-B) y (B-A).

Cuando los dos valores A y B son iguales el algoritmo finaliza. Así, podemos tomar el resultado MCD de cualquiera de los dos registros A o B.

Unidad de Control.

A continuación definimos las entradas y salidas de la Unidad de Control (ver Figura 8). Este bloque toma como entradas las señal de inicio (**Ini**) y los resultados de la comparación de los dos números (**A<>B** y **A>B**). Como salida deberá generar las señales de control para determinar qué canales se seleccionan en los multiplexores (**Mx1**, **Mx2**, **Mx3** y **Mx4**), la carga de los registros (**LdA** y **LdB**), y la señal que indicará que el algoritmo se ha completado (**Fin**).

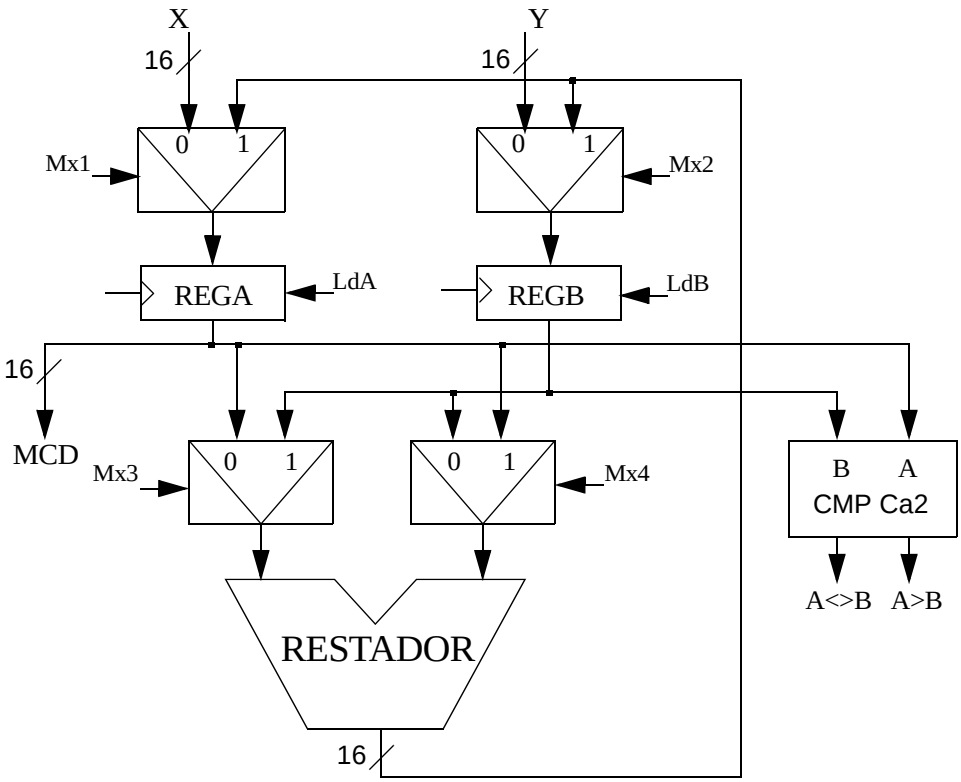


Figura 6. Unidad de proceso para el Máximo Comun Divisor.

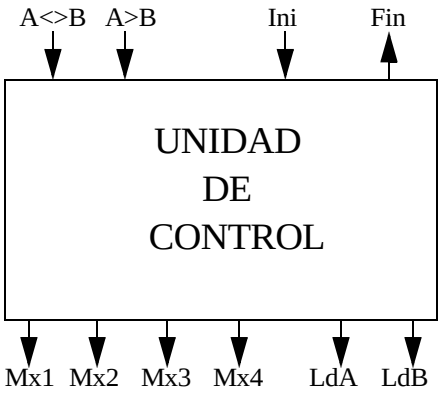
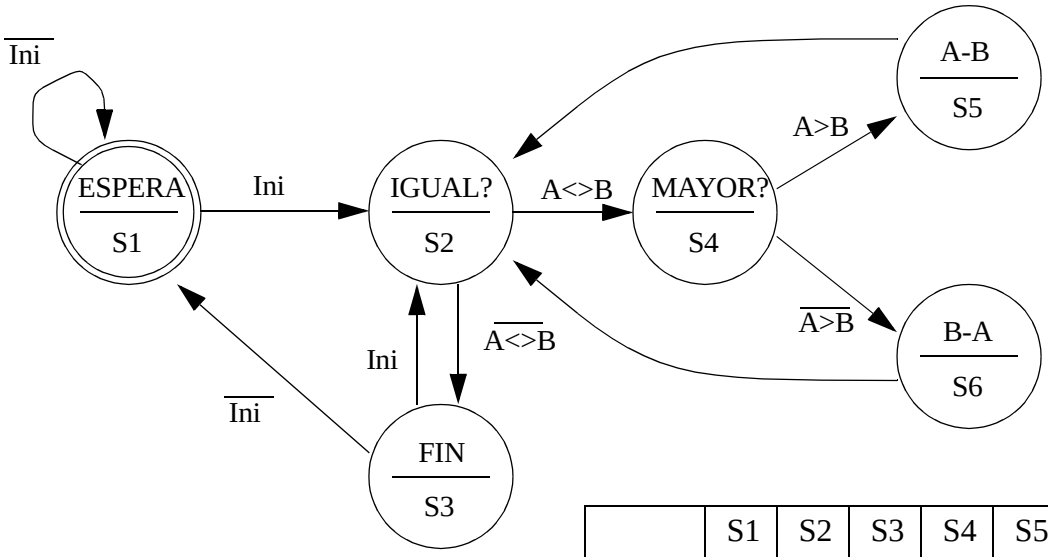


Figura 7. Unidad de control para el Máximo Comun Divisor.

Una vez determinadas las entradas y salidas de la Unidad de Control podemos pasar a su diseño mediante un Modelo de Moore (ver Figura 9).

En este sistema utilizamos un estado inicial (**ESPERA**) en el cual esperamos la llegada de una nueva pareja de datos X e Y (cuando la señal Ini se activa). En este estado es preciso almacenar los datos recién llegados en los registros para lo cual activamos las señales de carga (LdA y LdB) y seleccionamos los canales adecuados (Mx1 = Mx2 = 0).



	S1	S2	S3	S4	S5	S6
Fin	0	0	1	0	0	0
Mx1	0	X	0	X	1	X
Mx2	0	X	0	X	X	1
Mx3	X	X	X	X	0	1
Mx4	X	X	X	X	0	1
LdA	1	0	1	0	1	0
LdB	1	0	1	0	0	1

Figura 8. Detalle del modelo de Moore para la Unidad de Control.

Una vez tenemos los dos datos pasamos a un estado en el cual comparamos si los dos números son iguales (**IGUAL**). Esta información la podemos conseguir del comparador (salida A<>B). En este estado no se modifican los registros de la Unidad de Proceso ya que sólo se utiliza la información generada por el comparador.

Si los dos números son iguales el algoritmo debe finalizar, pasando al estado **FIN**. En este estado activamos la señal de finalización (Fin=1) y esperamos la llegada de nuevos datos. Para ello volvemos a activar las señales de load de los registros. Si mientras estamos en este estado llegan nuevas entradas pasaremos al estado de comparación de igualdad. En caso de que no lleguen nuevas entradas continuaremos esperando en el estado de **ESPERA** ya que debemos desactivar la señal de Fin.

Si los datos no son iguales, pasamos del estado **IGUAL?** al estado **MAYOR?**, en el que comparamos si A es mayor que B. Si es así pasaremos al estado **A-B** para actualizar el contenido del registro **REGA**. En caso contrario (A<B) debemos pasar al estado **B-A** donde actualizaremos el registro **REGB**.

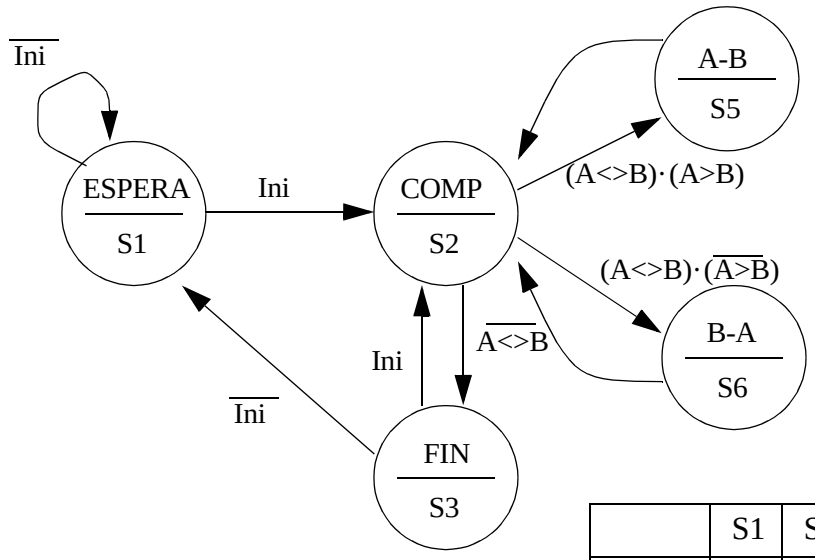
Para actualizar los registros es necesario realizar la resta correspondiente (A-B o bien B-A). Podemos realizar A-B si activamos el canal 0 en los multiplexores (Mx3 = Mx4 = 0), mientras que B-A se realiza activando el canal 1 de los multiplexores (Mx3 = Mx4 = 1). La actualización de los registros REGA y REGB se realiza seleccionando el canal 1 de los multiplexores (Mx1 = Mx2 = 1) y activando el load del registro adecuado en cada estado (LdA=1 en MAYOR y LdB=1 en MENOR).

Una vez completadas estas operaciones se ha terminado una iteración del algoritmo. Por tanto debemos continuar con la nueva pareja de datos A y B, comparado si son iguales en el estado **IGUAL**.

Optimización de la Unidad de Control.

Tras completar el diseño, y a modo de curiosidad, ya que no es un objetivo de esta asignatura que contruyáis grafos con el número mínimo de estados, podemos observar que todavía es posible realizar alguna Optimización adicional de la Unidad de Control. Esta optimización reducirá el número total de ciclos requeridos en el cálculo del MCD (ver Figura 10).

Una posible Optimización consiste en ver que los estados de comparación de **IGUAL?** y **MAYOR?** pueden reducirse a un solo estado que denominaremos **COMP**. Esto es posible ya que el comparador realiza su función durante un ciclo (estado **IGUAL?**) y mantiene su salida en el siguiente (estado **MAYOR?**). Además, es importante darse cuenta de que las tres posibles opciones (A=B, A>B y A<B) son excluyentes.



	S1	S2	S3	S5	S6
Fin	0	0	1	0	0
Mx1	0	X	0	1	X
Mx2	0	X	0	X	1
Mx3	X	X	X	0	1
Mx4	X	X	X	0	1
LdA	1	0	1	1	0
LdB	1	0	1	0	1

Figura 9. Detalle del modelo de Moore para la Unidad de Control después de fundir los estados IGUAL y MAYOR.

ENUNCIADOS DE PROBLEMAS

Todos los sistemas que hay que diseñar en los problemas enunciados a continuación, excepto que se indique lo contrario, tienen una entrada de control de 1 bit denominada *Ini* que valida la información disponible en las entradas de datos. Asimismo, el sistema generará una señal de salida *Fin* que valida los resultados de la salida de datos. El funcionamiento es similar al ejemplo del sumador de 4 números que hemos visto anteriormente.

La UP se diseñará con bloques combinacionales y secuenciales como los del chuletario. La UC se especificará mediante un grafo de estados y no se implementará, excepto que se indique expresamente.

En todos los problemas, aunque no se indica en los enunciados, debe calcularse la cota inferior del tiempo de ciclo para que el sistema funcione correctamente.

PROBLEMA 1.
Diseñar un circuito que obtenga el máximo valor de entre *n* números naturales que llegan al sistema ciclo a ciclo. En el primer ciclo, cuando la entrada de control *Ini* vale 1, en la única entrada de datos del sistema está disponible el valor *n*. En los siguientes *n* ciclos estarán disponibles, en esa misma entrada, los valores de los *n* números a razón de uno por ciclo. El valor máximo debe estar disponible en la salida *RESUL* cuando finalice el cálculo. La salida se validará poniendo a 1 la salida de control *Fin*. Más formalmente:

MÓDULO "MÁXIMO DE N NÚMEROS"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	8 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = \underset{i=1}{\overset{DATO(c1)}{MAXIMO}} \{DATO(c1+i)\}$$

$$Fin(c2) = 1$$

siendo *DATO(c1 + i)*, para *0 <= i <= DATO(c1)*, un número natural con rango [0,255] codificado en binario con 8 bits.

FIN MÓDULO

PROBLEMA 2.
Diseñar un sistema que cuente el número de unos presentes en una palabra de 8 bits. La palabra está disponible en la entrada *DATO* durante el ciclo en que la señal *Ini* vale 1. El resultado debe estar disponible en la salida *RESULT,* de 4 bits, durante un ciclo cuando lo indique la señal *Fin*. Este sistema se puede especificar más formalmente como sigue:

MÓDULO "CONTADOR DE UNOS"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	4 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = \sum_{i=0}^{8} DATO(c1)[i]$$

$$Fin(c2) = 1$$

donde *DATO(c1)[i]* es el bit con peso 2^i de *DATO(c1)*.

FIN MÓDULO

PROBLEMA 3.
Diseñar un sumador de números naturales codificados en binario con 16 bits usando un único sumador de 8 bits. Indicar cuándo se produce overflow en el resultado.

MÓDULO "SUMA EN DOBLE PRECISIÓN"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO1, DATO2	16 bits
	de control:	Ini	1 bit
Salidas	de datos:	SUMA	16 bits,
		OVERFLOW	1 bit
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$SUMA(c2) = (DATO1(c1) + DATO2(c1)) mod 2^{16}$$

$$OVERFLOW(c2) = INT((DATO1(c1) + DATO2(c1))/2^{16})$$

$$Fin(c2) = 1$$

donde *DATO1(c1), DATO2(c1), SUMA(c2)* son números naturales con rango $[0,...,2^{16} -1]$ codificados en binario y *OVERFLOW(c2)* es un dígito binario.

FIN MÓDULO

PROBLEMA 4.
Modificar el circuito del ejercicio 3 para que mantenga la misma funcionalidad, pero ahora los números a sumar y el resultado son enteros codificados en complemento a 2 con 16 bits.

PROBLEMA 5.
Diseñar un sumador en el que los 2 números a sumar y el resultado son enteros con rango [-127,...,127] codificados en signo y magnitud con 8 bits (el bit de más a la izquierda es el bit de signo: 0 positivo y 1 negativo). En la UP habrá solamente 1 ALU, además de los multiplexores y los registros de carga paralela que sean necesarios.

PROBLEMA 6.
Diseñar el siguiente sistema:

MÓDULO "FA DECIMAL"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO1, DATO2	4 bits
	de control:	Ini	1 bit
Salidas	de datos:	SUMA	4 bits,
		CARRY	1 bit
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$CARRY(c2) = INT((DATO1(c1) + DATO2(c1))/10)$$

$$Fin(c2) = 1$$

$$SUMA(c2) = (DATO1(c1) + DATO2(c1))mod10$$

donde *DATO1(c1), DATO2(c1), SUMA(c2)* son dígitos decimales (con rango [0,...,9]) codificados en binario con 4 bits (BCD) y CARRY(c2) es un dígito binario.

FIN MÓDULO

PROBLEMA 7.
Diseñar el siguiente sistema, utilizando en la UP solamente 1 ALU y otros bloques.

MÓDULO "CÁLCULO"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO1, DATO2	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	8 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$Fin(c2) = 1$$

$$RESULT(c2) = INT(0{,}75 \times MAX(DATO1(c1), DATO2(c1)))$$

donde *DATO1(c1), DATO2(c1), RESULT(c2)* son números naturales con rango [0,...,255] codificados en binario con 8 bits.

FIN MÓDULO

Nota: puede ser de ayuda una de las siguientes igualdades: 0.75 = 1/2 + 1/4, 0.75 = 3/4, 0.75 = 1 - 1/4.

PROBLEMA 8.

Diseñar el siguiente sistema, utilizando en la UP una ALU y otros bloques.

MÓDULO "CUADRADO"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	16 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = DATO(c1)^2$$

$$Fin(c2) = 1$$

donde *DATO(c1)* es un número natural con rango 0 hasta 255 codificado en binario con 8 bits y *RESULT(c2)* es un número natural con rango [0,...,65535] codificado con 16 bits.

FIN MODULO

Nota: $X^2 = \sum_{i=0}^{X-1} X$

PROBLEMA 9.

Diseñar un circuito capaz de multiplicar dos números enteros X e Y codificados en complemento a 2 con 8 bits cada uno, utilizando para la UP solamente 1 ALU de 8 bits, multiplexores, registros y puertas. Se puede utilizar el siguiente algoritmo:

1. Obtener los valores absolutos de X y Y: |X|, |Y|
2. Calcular |X|*|Y|
3. Cambiar de signo |X|*|Y| en caso necesario

MÓDULO "MULTIPLICADOR CA2"

ENTRADAS/SALIDAS

Entradas	de datos:	DATO1, DATO2	8 bits
	de control:	Ini	1 bit
Salidas	de datos:	RESULT	16 bits
	de control:	Fin	1 bit

FUNCIONAMIENTO

si *Ini(c1) = 1* entonces

$$RESULT(c2) = DATO(c1) \times DATO2(c1)$$

$$Fin(c2) = 1$$

donde *DATO1(c1), DATO2(c1)* son números enteros con rango [-128,...,127] codificados en complemento a 2 con 8 bits y *RESULT(c2)* es un entero con rango [-163846,...,16383] codificado en complemento a 2 con 16 bits.

FIN MÓDULO