

## CONTROLADORES

En el Capítulo 7 introdujimos algunas ideas útiles e importantes relativas a los sistemas lógicos secuenciales, es decir, sistemas con memoria. Para ilustrar el diseño de dichos sistemas empleamos el ejemplo del detector de secuencia. Un tipo de sistema secuencial que tiene un rango mayor de aplicabilidad es el controlador. Los controladores son sistemas secuenciales que suministran niveles lógicos apropiados en los tiempos apropiados para controlar una secuencia de operaciones lógicas sencillas que, en conjunto, realizan una operación complicada. Consideraremos en primer lugar cómo pueden realizarse estas operaciones lógicas sencillas.

## 8.1 TRANSFERENCIAS DE REGISTROS

Las operaciones elementales, lógicas o aritméticas que pueden realizarse sobre palabras lógicas son bastante simples. Una operación básica típica y la más importante consiste en transferir simplemente el contenido de un registro a otro. Un mecanismo para esta transferencia se muestra en la figura 8.1-1, donde tenemos dos registros, el  $A$  y el  $B$ . Los registros considerados aquí constan de un array de varios cerrojos estáticos set-reset (RS) como en la figura 4.2-1. Solamente se muestran un cerrojo del registro  $A$ , el cerrojo  $A_i$ , y otro del registro  $B$ , el  $B_i$ . Los demás cerrojos están representados por los puntos suspensivos que aparecen a ambos lados de dichos cerrojos. Se interponen dos pueras AND entre los cerrojos, las cuales pueden habilitarse poniendo el terminal de control marcado «Mover A a B» al nivel lógico 1. En esta habilitación  $S$  de  $B_i$  asumirá el valor  $Q$  de  $A_i$  y  $R$  el valor  $\bar{Q}$  de  $A_i$ . El cerrojo  $B_i$  asumirá además el estado de  $A_i$ ; cuando el terminal de control «Mover A a B» vuelva al nivel lógico 0,  $R$  y  $S$  se harán 0 y el estado de  $A_i$  quedará encerrado en  $B_i$ . En resumen, poniendo el terminal de control en su nivel activo se genera una orden a la que responde el circuito. El terminal de control a menudo se denomina terminal de orden. Notar que en este proceso

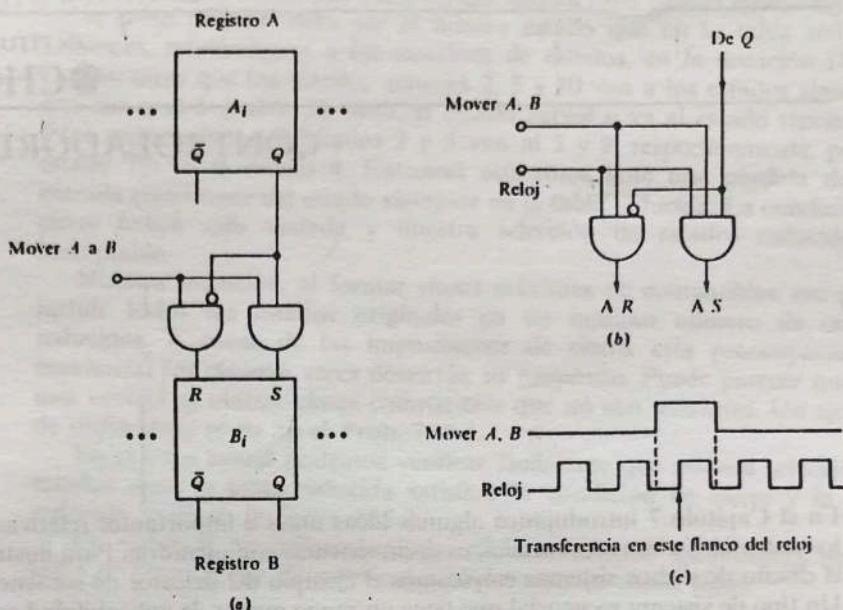


Figura 8.1-1 (a) La puesta a 1 lógico, brevemente, del terminal de control «Mover A a B» transmite el contenido del registro A al B. (b) Se añade un reloj para sincronizar la operación. (c) Diagrama de tiempo de la operación de transmisión.

la palabra almacenada en el registro A permanecerá inalterada pero se perderá, por supuesto, cualquier palabra almacenada en el registro B antes de la transferencia.

Si, a título de ejemplo, los registros A y B son de 16 bits, entonces se requieren 16 líneas de conexión, así como 16 pares de puertas AND. La inversión realizada antes de la puerta AND que precede a la entrada R sería innecesaria si hubiésemos añadido una segunda línea de conexión de la salida  $\bar{Q}$  de  $A_1$ . No lo hemos hecho así en base a que físicamente la implementación de una inversión es a menudo más fácil que añadir una conexión.

Lo más importante, indicamos que transferir el contenido del registro A al B requiere sencillamente que habilitemos brevemente un array de puertas que cambien el nivel lógico de un terminal de control. Si tratamos con un sistema sincrónico (el caso usual), podríamos querer que la transferencia se haga en sincronismo con el reloj. Esta transferencia sincrónica puede realizarse añadiendo una entrada de reloj a las puertas AND, como indica la figura 8.1-1b. La figura 8.1-1c muestra una señal de reloj y un cambio del nivel de habilitación en «Mover A a B» que selecciona un ciclo de reloj particular durante el que ocurre la transferencia.

Supongamos que tenemos muchos registros y necesitamos la posibilidad de efectuar transferencias de un registro a otro. Si realizásemos conexiones

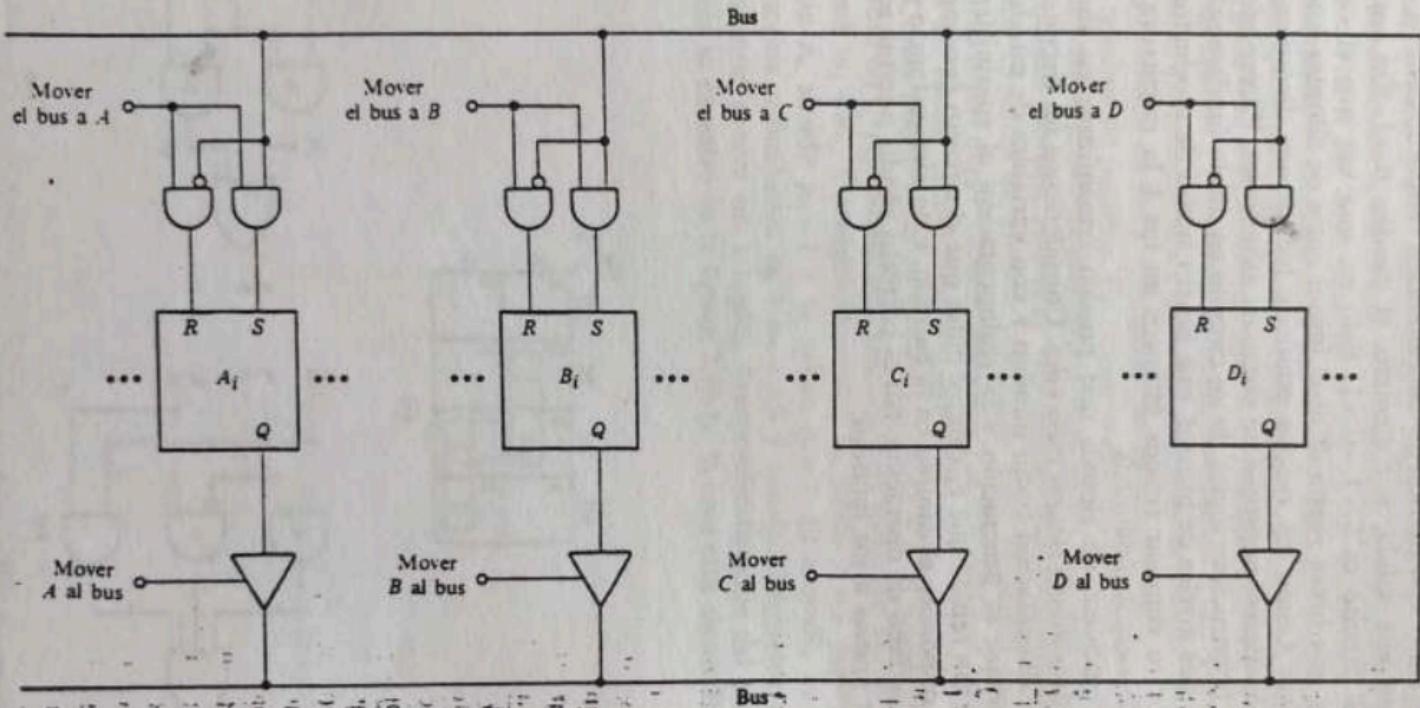


Figura 8.1-2 Registros compartiendo un bus común. Al alcanzar en un instante el 1 lógico una entrada y una salida de control, se realiza la transferencia entre registros.

individuales de cada registro a todos los demás, el número de éstos podía ser enorme. En este caso, podríamos utilizar un esquema de multiplexación (ver sección 3.19) en el que se emplee un bus común. Dicho bus se muestra en la figura 8.1-2 y sirve de interconexión entre muchos registros. Solamente se indican cuatro registros y por cada uno de ellos sólo se muestra un cerrojo estático. Por consiguiente, se muestra un bus de un hilo. Si los registros fueran, por ejemplo, de 16 bits, habría 16 hilos separados en el bus. Como toda la información debe transferirse a un único bus, solamente se puede hacer cada vez una transferencia. Si, por ejemplo, quisieramos hacer una transferencia de *A* a *C* pondríamos al nivel de 1 lógico el terminal de control «Mover *A* al bus» y el estado del registro *A* se colocaría en el bus. (En la figura 8.1-2 hemos utilizado la conexión tres estados para acoplar las salidas de muchos registros a un único bus. Alternativamente, podría utilizarse la conexión colector-abierto de la sección 3.19.) Si ahora también ponemos en el nivel lógico 1 el terminal marcado «Mover el bus a *C*» se realizará la transferencia. Pero, de nuevo, el punto importante a señalar es que la operación de transferencia se realiza habilitando puertas, en este caso dos conjuntos de puertas, dejando disponibles las líneas sobre las que se cambia el nivel lógico al de habilitación.

## 8.2 OTRAS OPERACIONES

### Complementación

Una segunda operación lógica necesaria normalmente consiste en complementar una palabra. Aquí lo que hacemos es sustituir una palabra de un registro por una nueva palabra en la que cada bit es el complemento del bit correspondiente de la palabra original. La figura 8.2-1 muestra la palabra originalmente en el registro *A*; el complemento aparece en el registro *B*. La transferencia y complementación se realiza simplemente poniendo a 1 lógico los terminales de «Complemento» y «Mover *A* a *B*». Si se emplea reloj como se indicó, los terminales anteriores estarán en 1 lógico cuando el reloj también alcance el 1 lógico y la transferencia ocurrirá sincrónicamente con el flanco de reloj. En la figura 8.2-1a el registro *B* puede estar compuesto sencillamente por un array de cerrojos estáticos.

Supongamos, por otro lado, que no queremos utilizar un segundo registro en el que aparezca el complemento. Primero queremos utilizar sólo un registro y sustituir la palabra original por su complemento. Entonces nos proponemos leer del registro (para ver qué bit está en cada flip-flop) y, en el mismo ciclo de reloj, escribir en el flip-flop el complemento del bit leído. Para este propósito se requiere que los flip-flops del registro sean de un tipo especial (maestro-esclavo, etc., como se discutió en el Capítulo 4) que permitan lectura y escritura simultáneas. En el registro de la figura 8.2-1b utilizamos flip-flops *JK*. Cuando la línea «Complemento» esté en 1, el flip-flop comutará en la transición de disparo de reloj reemplazando de ese modo cada bit por su complemento.

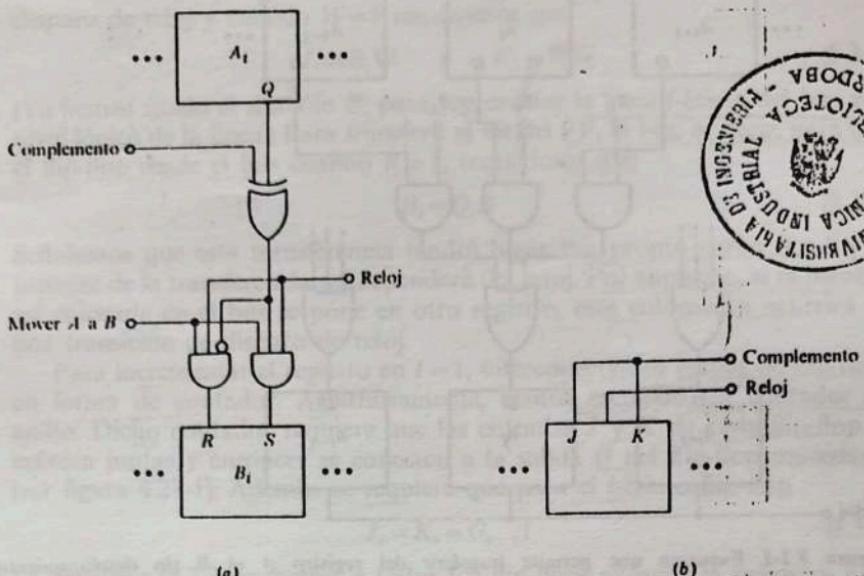


Figura 8.2-1 (a) Disposición que permite una transferencia del registro  $A$  al  $B$  del contenido de  $A$  (complemento = 0) o del complemento bit a bit de  $A$  (complemento = 1). (b) Esquema que permite la complementación sin transferencia.

## Desplazamiento

La figura 8.2-2 muestra una disposición que permite una transferencia del registro  $A$  al  $B$ . Dependiendo qué  $S_0$ ,  $S_L$  o  $S_R$  estén en 1 lógico el desplazamiento será directo o se realizará en una dirección u otra. Se muestran tres flip-flops consecutivos del registro  $A$  y otros tres del  $B$ . Se supone que el registro  $B$  está constituido por flip-flops tipo  $D$ . Hay una estructura de cuatro puertas (tres puertas AND y una OR) asociadas a la entrada de cada flip-flop del registro  $B$ , pero solamente dichas estructuras se muestran completamente. Si sólo  $S_0$  es  $S_0 = 1$ , el contenido de  $A_i$  se transferirá a  $B_i$  en la ocurrencia de un flanco de disparo de reloj. Si sólo  $S_L$  es  $S_L = 1$  habrá un desplazamiento a la izquierda y transferencia.  $S_R = 1$  conllevará desplazamiento a la derecha y transferencia. Por supuesto, en un desplazamiento a la izquierda necesitaremos hacer provisión especial para el bit que va a ser desplazado en el flip-flop de más a la derecha, ya que este bit debe ser suministrado por una fuente externa. Un comentario similar se aplica al bit que será desplazado al flip-flop de más a la izquierda, en el desplazamiento a la derecha. En cualquier situación, y lo más importante, señalamos que la operación a realizar se completará manteniendo en 1 lógico, durante cada ciclo de reloj, uno cualquiera de los terminales de control  $S_0$ ,  $S_L$  o  $S_R$ .

Supongamos que no necesitamos la facultad de transferir y desplazar una palabra, sino únicamente de desplazarla manteniéndola en el mismo registro.

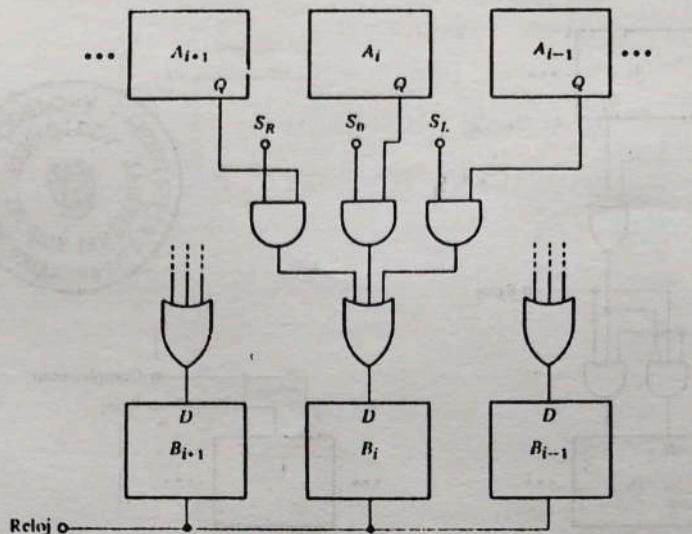


Figura 8.2-2 Esquema que permite transferir del registro  $A$  al  $B$  sin desplazamiento ( $S_RS_0S_L=010$ ), con desplazamiento a la izquierda ( $S_RS_0S_L=001$ ) o con desplazamiento a la derecha ( $S_RS_0S_L=100$ ).

Entonces únicamente necesitaríamos un registro de desplazamiento a la izquierda y derecha, ya descrito en la sección 4.18.

### Incrementación y decrementación

A menudo debemos almacenar un número en un registro e incorporarle a éste la facultad de cambiar el número almacenado en  $+1$  o  $-1$ . Estas operaciones se denominan incrementación y decrementación. Un registro que responda a la orden de cambio de cuenta, se muestra en la figura 8.2-3, consta de un contador reversible (de cualquier tipo de rizado o sincrónico) con la modificación que el reloj no se aplica directamente a la entrada de reloj del contador. Dicha señal pasa a través de una puerta AND adicional. La otra

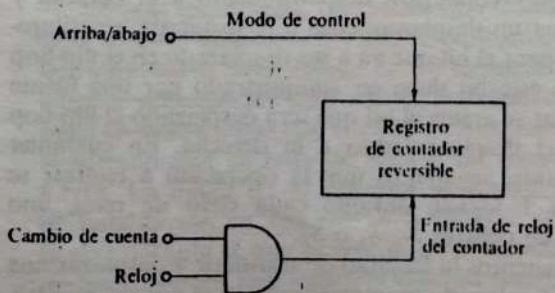


Figura 8.2-3 El registro contador incrementará o decrementará su cuenta dependiendo del nivel lógico del modo de control, si el terminal de «Cambio de cuenta» se mantiene en 1 lógico durante un ciclo de reloj.

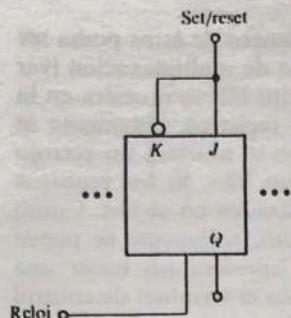


Figura 8.2-4 Flip-flop JK utilizado como parte de un registro y con la facultad de ser borrado o puesto en set.

entrada de dicha puerta es de habilitación y se denomina incremento. Si el terminal de cambio de cuenta se mantiene en 1 lógico durante un ciclo de reloj, el contador incrementará o decrementará su registro en 1, dependiendo del nivel lógico del modo de control.

#### Reset y set

Supongamos que necesitamos la facultad de borrar un registro (colocar  $Q$  de cada flip-flop en  $Q=0$ ) o poner en set un registro (cada  $Q=1$ ). Un flip-flop individual, tipo JK, de dicho registro se indica en la figura 8.2-4. Si la entrada set/reset está en 1 lógico, entonces  $J=1$  y  $K=0$ , así que en un flanco de disparo de reloj el flip-flop estará en set. Si set/reset está en 0 lógico, tendremos  $J=0$  y  $K=1$ , así que el flip-flop quedará en reset.

### 8.3 REGISTRO SENSIBLE A MÚLTIPLES ÓRDENES

Hemos visto que podemos construir registros que respondan a una u otra orden. La orden se transmite poniendo algún terminal de control en el nivel lógico que habilita una puerta o un array de puertas. En un sistema sincrónico que utilice reloj generalmente se realizará la orden en el flanco de disparo de una señal de reloj. Supongamos que en algún sistema digital necesitamos la facultad de ordenar una serie de operaciones. Entonces podemos tratar de realizar operaciones individuales en registros separados o construir un registro que sea capaz de responder a distintas órdenes. La primera alternativa ofrece la ventaja de la flexibilidad, la segunda nos puede permitir ahorrar hardware.

Como ejemplo de registro que pueda responder a varias órdenes diseñamos uno que responda a cinco órdenes. Estas cinco órdenes y sus símbolos asociados están listadas en la tabla 8.3-1. Así, nuestro registro tendrá cinco terminales de control  $W, R, I, C$  y  $Z$ . En cualquier instante, solamente uno de estos terminales estará en 1 lógico mientras los demás estarán en 0 lógico. Si, por ejemplo, tenemos  $W=1$ , entonces durante el flanco

Tabla 8.3-1 Órdenes a los cuales responde registro

Orden	Símbolo
1. Escribe en el registro la palabra del bus	$W$
2. Lee en el bus la palabra del registro	$R$
3. Incrementa el registro	$I$
4. Complementa el registro	$C$
5. Borra el registro para que sean cero todas las $Q$	$Z$

de disparo de la señal de reloj la palabra del bus se introducirá en el registro. Los  $n$  flip-flops del registro son  $FF_0, FF_1, \dots, FF_i, \dots, FF_{n-1}$ . En la figura 8.3-1 se muestra la línea  $B_i$  de las  $n$  líneas del bus, así como la lógica asociada del flip-flop  $FF_i$ . Aquí hemos decidido arbitrariamente usar flip-flops JK. Con el

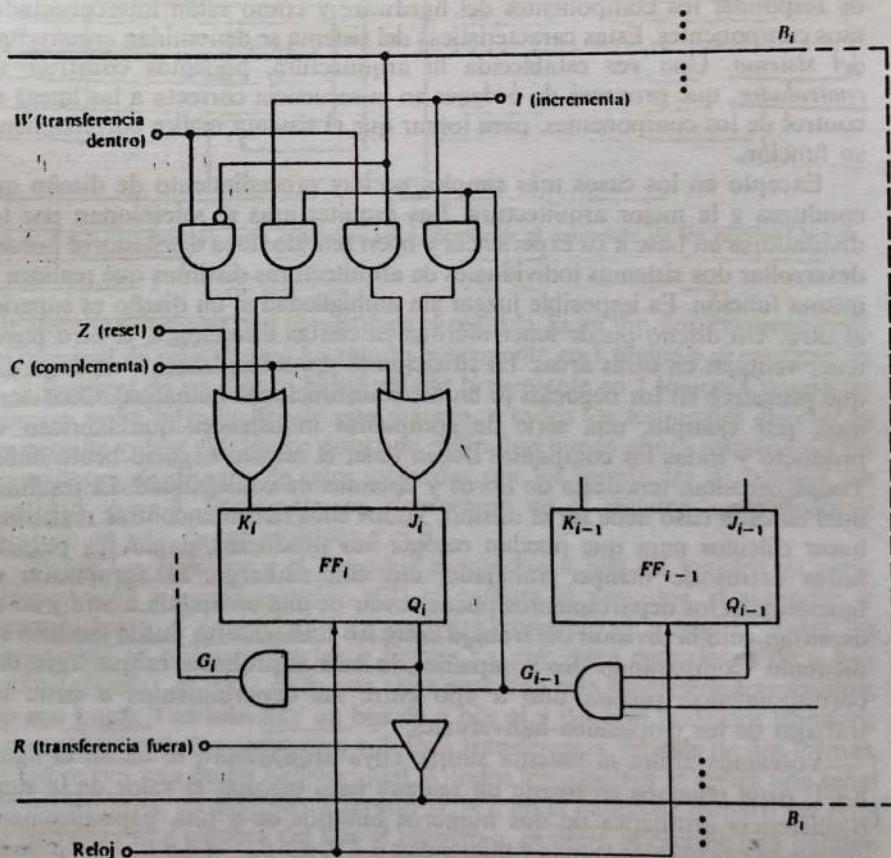


Figura 8.3-1 Una de las etapas de un registro que responderá a cinco órdenes.

fin de introducir la lógica de la línea del bus  $B_i$  en  $FF_i$  en una transición de disparo de reloj y cuando  $W=1$  requerimos que

$$J_i = B_i W \quad y \quad K_i = \bar{B}_i W \quad (8.3-1)$$

(Ya hemos usado el símbolo  $B_i$  para representar la línea  $i$ -ésima del bus y el nivel lógico de la línea.) Para transferir el bit del  $FF$  al bus, es decir, para leer el flip-flop desde el bus cuando  $R=1$ , requerimos que

$$B_i = Q_i R \quad (8.3-2)$$

Señalamos que esta transferencia tendrá lugar tan pronto como  $R=1$  y el instante de la transferencia no dependerá del reloj. Por supuesto, si la palabra así colocada en el bus se pone en otro registro, esta colocación ocurrirá en una transición de disparo de reloj.

Para incrementar el registro en  $I=1$ , interconectamos etapas de flip-flops en forma de contador. Arbitrariamente, hemos escogido un contador de anillo. Dicho contador requiere que las entradas  $J$  y  $K$  de cada flip-flop se encuenen juntas y entonces se conecten a la salida  $\bar{Q}$  del flip-flop precedente (ver figura 4.27-1). Además se requiere que para el  $i$ -ésimo flip-flop

$$J_i = K_i = G_{i-1} I \quad (8.3-3)$$

La ecuación (8.3-3) se aplica a todos los flip-flops excepto al primero,  $FF_0$ , que no tiene etapa precedente. En este caso especial se requiere

$$J_0 = K_0 = 1 \cdot I \quad (8.3-4)$$

Para que un flip-flop complemente, es decir, commute, cuando  $C=1$ , ponemos

$$J_i = K_i = C \quad (8.3-5)$$

Finalmente para hacer borrar el flip-flop cuando  $Z=1$  pero sin que esté afectado por  $Z$  cuando  $Z=0$  fácilmente verificamos en la tabla de verdad de la figura 4.11-1 que se requiere

$$K_i = Z \quad J_i = 0 \quad (8.3-6)$$

En resumen, teniendo en cuenta que en cualquier instante solamente una de las variables de control  $W$ ,  $R$ ,  $I$ ,  $C$  y  $Z$  está en 1 lógico tenemos de las ecuaciones (8.3-1) a (8.3-3), (8.3-5) y (8.3-6) que:

$$J_i = B_i W + G_{i-1} I + C \quad K_i = \bar{B}_i W + G_{i-1} I + C + Z \quad (8.3-7)$$

y

$$B_i = Q_i R \quad (8.3-8)$$

El primer flip-flop  $FF_0$  es especial en el sentido que los términos  $G_{i-1} I$  de la ecuación (8.3-7) son reemplazados simplemente por los términos  $1 \cdot I = I$ , como se indicó en la ecuación (8.3-4).

En la figura 8.3-1 se muestran las puertas asociadas al  $FF_i$  representadas

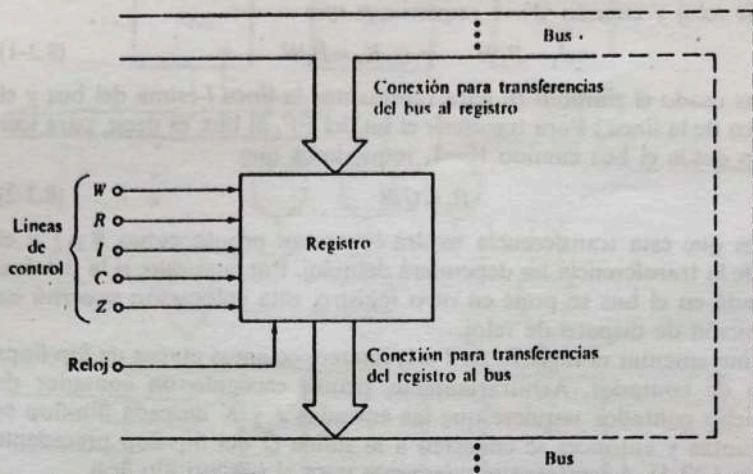


Figura 8.3-2 Representación funcional del registro cuyo único estado se da en la figura 8.3-1.

por las ecuaciones (8.3-7) y (8.3-8). Por supuesto que se requiere una estructura de puertas similar (no mostradas) para cada flip-flop del registro.

En la figura 8.3-2 se da un diagrama de bloques del registro mostrando sus conexiones al bus. Se indican las cinco líneas de las órdenes de control así como la entrada de reloj.

#### 8.4 UN SENCILLO CONTROLADOR

Hemos visto que es factible construir de forma correcta registros y circuitos combinacionales que nos permitan realizar sobre palabras sencillas operaciones lógicas o aritméticas y almacenar los resultados. Una vez construido el hardware adecuado, la operación se efectúa sin más que poner a 1 lógico (o, si nos place, a 0 lógico) la linea de control que nos sirva para habilitar una puerta o un array de puertas. Por este acto de habilitación se da una orden a la que responde el hardware. Las operaciones elementales cuya implementación hardware hemos examinado incluyen transmisión a y de un bus, incrementación, complementación y desplazamiento. De forma similar podemos construir hardware para realizar otras operaciones elementales. Supongamos, por ejemplo, que un registro de  $n$  bits que contenga la palabra  $R_{n-1}, \dots, R_1, \dots, R_0$  acepta la palabra de entrada  $A_{n-1}, \dots, A_i, \dots, A_0$  y cambia su registro a  $R'_{n-1}, \dots, R'_1, \dots, R'_0$ , donde cada nuevo bit es  $R'_i = R_i A_i$ . Dicho registro realiza la operación AND. Registros que respondan a órdenes para realizar otras operaciones lógicas son igualmente posibles. Así podemos tener  $R'_i = R_i + A_i$ ,  $R'_i = R_i \oplus A_i$ , etc. (ver Probs. 8.2-1, 8.2-2).

Si necesitamos realizar una serie de operaciones lógicas y aritméticas diferentes sobre palabras, podemos construir un registro que pueda ordenar la realización de todas las operaciones elegidas. Por ejemplo, podríamos

diseñar un registro que permita transferir a y de un bus, complementar, incrementar, desplazar, borrar la AND-lógica, la OR-lógica, etc., o podríamos decidir utilizar una serie de registros que individualmente permitan pocas operaciones, pero que entre todos puedan realizar cualquier operación requerida. En el primer caso se requerirán pocas transferencias registro-a-registro. En el último caso habrá mayor número de registros que individualmente serán más simples y el sistema se puede prestar más fácilmente a modificaciones. (Dichos registros, como hemos descrito, se denominan registros de trabajo en contraste con los registros de almacenamiento, que sirven solamente para almacenar una palabra.) Alternativamente, podemos decidir utilizar registros de almacenamiento mejor que de trabajo empleando circuitos combinacionales que responden a órdenes, como la ALU descrita en la sección 5.13. En cualquier circunstancia, el comienzo de diseño de un sistema digital comienza con una decisión (al menos tentativa) de los componentes del hardware que se van a emplear, a qué órdenes son capaces de responder los componentes del hardware y cómo están interconectados esos componentes. Estas características del sistema se denominan arquitectura del sistema. Una vez establecida la arquitectura, podemos construir un controlador, que proveerá de órdenes en la secuencia correcta a las líneas de control de los componentes, para lograr que el sistema realice correctamente su función.

Excepto en los casos más simples no hay procedimiento de diseño que conduzca a la mejor arquitectura. Las arquitecturas se seleccionan por los diseñadores en base a su experiencia y buen sentido. Dos diseñadores pueden desarrollar dos sistemas individuales de arquitecturas distintas que realicen la misma función. Es imposible juzgar sin ambigüedad si un diseño es superior al otro. Un diseño puede tener méritos en ciertas direcciones, el otro puede tener ventajas en otras áreas. La situación es aproximadamente análoga a la que prevalece en los negocios (o en otras instituciones humanas). Consideremos, por ejemplo, una serie de compañías industriales que fabrican un producto y todas las compañías deben tener el mismo negocio bruto anual. Todas necesitan teneduría de libros y sistemas de contabilidad. El resultado final en cada caso debe ser el mismo. Todos ellos deben encontrar registros y hacer cálculos para que puedan recoger sus productos, pagar sus pagares, horas extras de tiempo trabajado, etc. Sin embargo, la agrupación de funciones en los departamentos puede variar de una compañía a otra y en un departamento la división del trabajo entre los trabajadores puede también ser diferente. Comparando dos compañías, lo más improbable es que haya una correspondencia precisa uno a uno entre los departamentos o entre los trabajos de los empleados individuales.

Volvamos ahora al sistema simple cuya arquitectura se da en la figura 8.4-1. Aquí tenemos en mente un sistema para calcular el valor de la suma o diferencia aritmética de dos números binarios de  $n$  bits. Específicamente queremos calcular las sumas y diferencias  $\alpha + \beta$ ,  $\alpha - \beta$ ,  $-\alpha + \beta$  y  $-\alpha - \beta$  de los números en los registros  $\alpha$  y  $\beta$ . El mecanismo por el que estos números se introducen en los registros  $\alpha$  y  $\beta$  no se indica en la figura. Supongamos que se introducen asincrónicamente por los terminales directos de set y reset de los

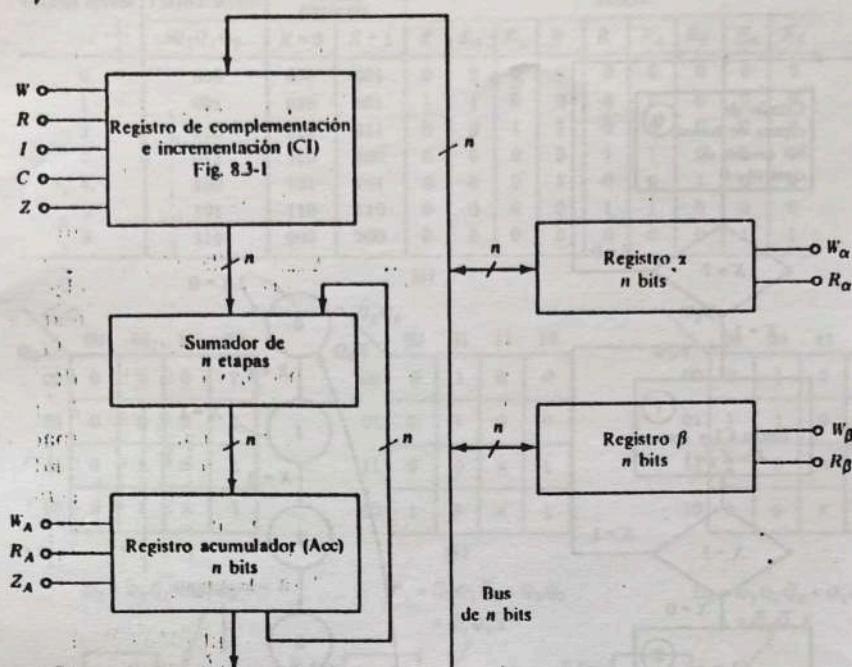


Figura 8.4-1 Arquitectura para combinar aritméticamente el contenido de los registros  $\alpha$  y  $\beta$ .

flip-flops individuales que constituyen el registro. Si un flip-flop contiene un 0, su terminal de reset directo ha estado brevemente en 1 lógico y si contiene un 1, el terminal de set directo habrá estado brevemente en 1 lógico. Cuando un número se ha introducido de esta manera, a todos los terminales directos se les permite volver al 0 lógico para que el flip-flop pueda ahora responder a las transiciones de disparo de reloj utilizadas por los terminales de control. La cantidad  $+ \alpha + \beta$ , finalmente, se almacenará en el registro acumulador y también en uno de los registros  $\alpha$  o  $\beta$ .

Todos los registros y el sumador acomodan  $n$  bits. El registro de complementación e incrementación (CI) se conecta al sumador, el sumador al acumulador y el acumulador se vuelve a conectar al sumador, todas las conexiones se realizan con las  $n$  líneas. Ya que estas conexiones de  $n$  bits están dedicadas, es decir, cada una sirve para una sola función de transmisión, no son buses. También hay un bus de  $n$  bits al y del cual podemos transferir los contenidos de los registros  $\alpha$  y  $\beta$ . Esta transferencia, factible de dos formas, se indica por una flecha bidireccional. A todos los registros se aplica una señal de reloj común, no indicada. Cuando  $W_\alpha = 1$ , en el flanco de disparo del reloj se transfiere una palabra del bus al registro  $\alpha$ ; es decir, se escribe una palabra en el registro. Cuando  $R_\alpha = 1$ , se lee una palabra del registro al bus. Comentarios similares se aplican a la respuesta del registro  $\beta$  a  $W_\beta$  y  $R_\beta$ .

Cuando  $W_A = 1$ , la salida del sumador se registrará en el acumulador y cuando  $R_A = 1$ , el contenido del acumulador se colocará en el bus. Los contenidos del acumulador están permanentemente conectados a las conexiones de los  $n$  bits que vuelven al sumador. No hay control sobre esa conexión. El acumulador se borra en la transición de disparo de reloj si  $Z_A = 1$ .

Consideremos ahora cómo puede gobernarse el sistema de la figura 8.4-1 para calcular  $\alpha + \beta$  y almacenar el resultado en el registro  $\alpha$ . Se necesitan una serie de operaciones elementales, denominadas *microoperaciones*, cada una de las cuales requiere un ciclo de reloj. La secuencia se da en la tabla 8.4-1.

Señalemos que en esta secuencia de operaciones elementales tanto  $\alpha$  como  $\beta$  pasan a través de CI. Por supuesto, no hay necesidad de hacerlo así, y el tiempo de dos ciclos de reloj se ha perdido. Aunque a causa de la arquitectura de nuestro sistema no lo hubiéramos escogido, si hubiésemos comenzado con una arquitectura más elaborada, que suministrase acceso directo al sumador, habríamos podido evitar el registro CI. Notar que no es necesario borrar el registro CI.

Ahora diseñaremos un controlador que fabricará los niveles lógicos requeridos para dividir la secuencia de nuestra máquina en sus pasos. Es claro que el controlador sea una máquina secuencial con seis estados, ya que han de realizarse seis operaciones separadas. Sin embargo, en el caso presente encontramos una situación que no aparecía en el Capítulo 7 cuando diseñábamos los detectores de secuencia. En un detector de secuencia se permitía que la máquina funcionase continuamente, ya que considerábamos que la secuencia de entrada era también continua. En el caso presente queremos detener la máquina después del último paso de la secuencia. De otra forma, la máquina continuará realizando su secuencia de operaciones. Después de una

Tabla 8.4-1

Ciclo de reloj	Líneas de control a poner en 1 lógico	Comentario
1	$Z_A$	Borra el registro acumulador de cualquier número que pueda haber quedado en una operación previa
2	$R_\alpha, W$	Lee $\alpha$ en el bus y escribe la palabra del bus al registro CI
3	$R, W_A$	Contenido de CI pasado por el sumador (la otra entrada del sumador es cero) y registrado en Acc
4	$R_\beta, W$	Contenido de $\beta$ transferido a CI
5	$R, W_A$	$\beta$ sumado al Acc
6	$R_A, W_\alpha$	Contenido del Acc transferido al registro $\alpha$

vuelta el nuevo contenido en el registro  $\alpha$  será  $\alpha' = \alpha + \beta$ . Despues de la siguiente vuelta tendremos  $\alpha'' = \alpha' + \beta$ , etc. Y si utilizamos un reloj con la frecuencia comunmente empleada en los sistemas electronicos digitales (100 KHz o mayor) no tendremos incluso tiempo de leer el registro  $\alpha$  antes de que este cambie.

Ademas, a los seis estados correspondientes a las seis microoperaciones sumemos un septimo en el que se detendrá y esperará el controlador cuando se haya completado la secuencia. Este estado extra nos dará tiempo para leer el resultado de nuestro cálculo y poner nuevos números en los registros  $\alpha$  y  $\beta$ . Consideramos entonces que al poner al controlador fuera del estado de espera hay una entrada  $X$  al controlador procedente de alguna fuente externa. Esta entrada puede suministrarse por un pulsador de botón. Cuando el botón se pulsa,  $X = 1$ ; en cualquier otro caso,  $X = 0$ . Cuando  $X = 0$ , el controlador una vez comenzado su secuencia continuará hasta que alcance el estado de espera. La siguiente secuencia no comenzará hasta que se haya pulsado el botón para que  $X$  se haga nuevamente  $X = 1$ . Sin embargo, como vemos ahora, no hemos resuelto completamente nuestro problema. Supongamos que habiendo pulsado el botón nos equivocamos soltándolo antes de que se haya completado la secuencia. Entonces el controlador puede pasar por varios ciclos antes de que, finalmente, llegue al reset en el estado de espera (a una velocidad de reloj 100 KHz, la secuencia de seis pasos se completa en solamente  $60 \mu$ ). En conjunto estamos conduciendo al controlador descrito por el diagrama de flujo de la figura 8.4-2a.

Hay siete estados que hemos numerado del 0 al 6. El número de estado se da en el círculo de la esquina superior derecha de las cajas de estados de la figura 8.4-2a. En el estado 0, el estado de espera, todas las entradas de control a la unidad aritmética de la figura 8.4-1 están en 0 lógico. Así, durante el tiempo que  $X = 0$  el controlador permanece en el estado 0. Cuando se pulsa el botón y  $X = 1$  el controlador va al estado 1, donde  $Z_A = 1$ , así que el registro acumulador se borra. El controlador no deja el estado 1 hasta que no se haya soltado el botón. En ese instante el controlador va al estado 2, donde  $R_2 = W = 1$  para que el contenido del registro  $\alpha$  desplace el registro  $C_1$ . Despues del estado 2, el progreso al estado 3 al resto de la secuencia se hace independientemente de si  $X = 0$  ó  $X = 1$ . El controlador eventualmente finaliza en el estado 0 y permanece en él hasta que se pulse de nuevo el botón. El diagrama de estados se da en la figura 8.4-2b.

## 8.5 IMPLEMENTACIÓN DEL CONTROLADOR

Ya que el controlador de la sección previa tiene siete estados, necesitaremos un circuito secuencial de tres flip-flops. Decidimos arbitrariamente utilizar flip-flops tipo D. La tabla de transición se da en la figura 8.5-1a. En la primera columna, para la identificación de estados, los hemos listado por su numeración en la figura 8.4-2a. Arbitrariamente hemos hecho la asignación de estados que se indica en la segunda columna. Sencillamente, hemos hecho, para cada estado, una asignación  $Q_2Q_1Q_0$  que cuando se lee como un

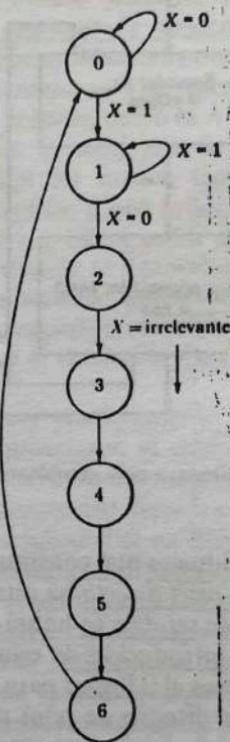
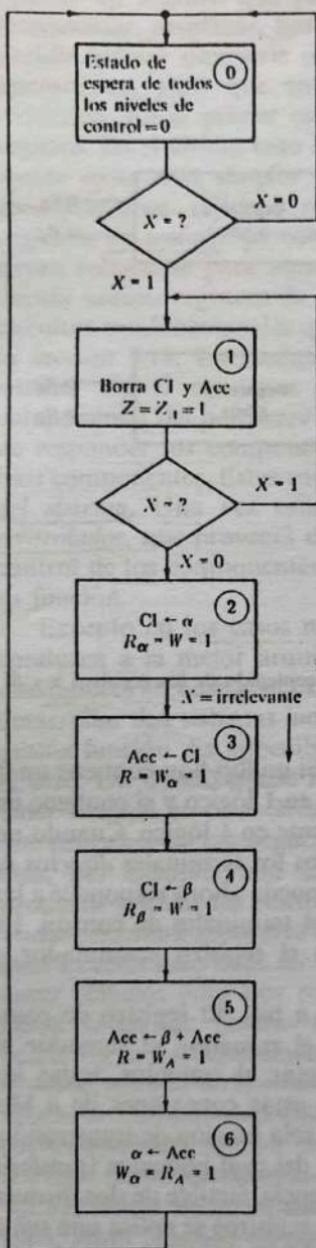


Figura 8.4-2 (a) Diagrama de flujo de un controlador que toma el sistema de la figura 8.4-1 a través de las microoperaciones requeridas para sumar los contenidos de  $R_\alpha$  y  $R_\beta$  y almacenar la suma en  $R_\alpha$ . (b) Diagrama de estados.

Estado actual	Estado actual	Estado siguiente		Salidas									
		$Q_2 Q_1 Q_0$	$X = 0$	$X = 1$	$Z$	$Z_A$	$R_\alpha$	$W$	$R$	$W_A$	$R_\beta$	$W_\alpha$	$R_A$
0	000	000	001		0	0	0	0	0	0	0	0	0
1	001	010	001		1	1	0	0	0	0	0	0	0
2	010	011	011		0	0	1	1	0	0	0	0	0
3	011	100	100		0	0	0	0	1	1	0	0	0
4	100	101	101		0	0	0	1	0	0	1	0	0
5	101	110	110		0	0	0	0	1	1	0	0	0
6	110	000	000		0	0	0	0	0	0	0	1	1

(a)

$Q_2 Q_1$	00	01	11	10
$Q_0 X$	00	0	0	1
	01	0	0	1
	11	0	1	X
	10	0	1	X

$Q_2 Q_1$	00	01	11	10
$Q_0 X$	00	0	1	0
	01	0	1	0
	11	0	0	X
	10	1	0	X

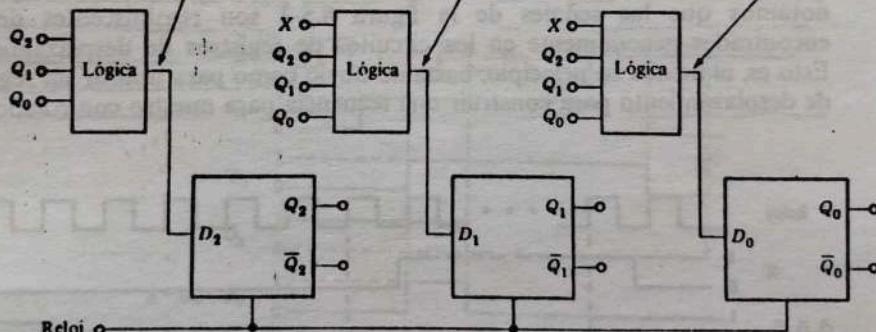
$Q_2 Q_1$	00	01	11	10
$Q_0 X$	00	0	1	0
	01	1	1	0
	11	1	0	X
	10	0	0	X

(b)

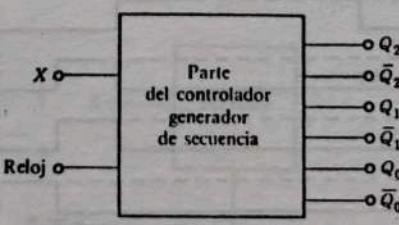
$$D_2 = Q_2 \bar{Q}_1 + Q_1 Q_0$$

$$D_1 = \bar{Q}_2 Q_1 \bar{Q}_0 + Q_2 Q_0 + \bar{Q}_1 Q_0 \bar{X}$$

$$D_0 = \bar{Q}_2 Q_1 \bar{Q}_0 + Q_2 \bar{Q}_1 \bar{Q}_0 + \bar{Q}_2 \bar{Q}_1 X$$



(c)



(d)

Figura 8.5-1 Diseño de un controlador cuyo diagrama de flujo y tabla de estados se dan en la figura 8.4-2: (a) tabla de transición; (b) diagrama K para las excitaciones de los flip-flops; (c) diagrama lógico para la parte de generación de secuencia del controlador, y (d) símbolo para el sistema lógico de (c).

número binario es igual al número de identificación decimal. Los siguientes estados, para  $X=0$  y  $X=1$  se toman directamente del diagrama de flujo o diagrama de estados de la figura 8.4-2.

El controlador es una máquina de Moore. Las salidas son completamente dependientes del estado del controlador, es decir, de los niveles lógicos en  $Q_2Q_1$  y  $Q_0$ . La única entrada  $X$  no tiene influencia directa en las salidas, ya que  $X$  sirve solamente para determinar si el controlador avanza por sus estados o si queda detenido. Las salidas listadas en las columnas restantes pueden leerse también directamente del diagrama de flujo.

En la figura 8.5-1b hemos construido diagramas K para las excitaciones  $D_2$ ,  $D_1$ ,  $D_0$  de los tres flip-flops. La lectura de cada diagrama se da directamente bajo el diagrama. En la figura 8.5-1c hemos dibujado el diagrama del circuito de la parte del controlador que determina el secuenciamiento de estado a estado. (La parte del controlador que genera las muchas salidas requeridas no se indica.) Las tres estructuras de puertas que generan las tres excitaciones  $D_2$ ,  $D_1$  y  $D_0$  no se dan explicitamente pero se indican por cajas rotuladas «lógica». Las entradas a las cajas lógicas son  $Q_2$ ,  $Q_1$  y  $Q_0$  y la variable  $X$ . Las salidas de las cajas lógicas se expresan en la figura como funciones booleanas. Por supuesto, si lo deseamos, podemos reemplazar las tres cajas lógicas usando puertas individuales por una ROM. Las entradas a la ROM, es decir, la dirección, serían entonces  $Q_2$ ,  $Q_1$ ,  $Q_0$  y  $X$  y las salidas, es decir, la palabra leída de la ROM sería  $D_2$ ,  $D_1$  y  $D_0$ . Finalmente, la parte generadora de secuencia del controlador (fig. 8.5-1c) se representa como un bloque en la figura 8.5-1d. Los únicos terminales explícitamente en evidencia son el reloj, la entrada  $X$ , y los terminales del flip-flop requeridos para implementar el decodificador que generará las salidas.

El decodificador que generará las salidas se da en la figura 8.5-2. A título de ejemplo, señalamos de la tabla de la figura 8.5-1a que  $Z$  y  $Z_A$  están en 1 lógico cuando y solamente cuando  $Q_2=0$ ,  $Q_1=0$  y  $Q_0=1$ . Por consiguiente, una puerta AND, con entradas como se indica, genera  $Z$  y  $Z_A$ . Señalamos que  $R$  y  $W_A$  están a 1 lógico cuando el estado es  $Q_2Q_1Q_0=011$  y también cuando  $Q_2Q_1Q_0=101$ . Aquí se requieren dos puertas AND y una puerta OR como se indica. El resto del decodificador se obtiene de la misma manera de la figura 8.5-1a que, en lo concerniente a las salidas, es una tabla de verdad donde se muestran las relaciones entre las salidas y  $Q_2Q_1$  y  $Q_0$ . Las figuras 8.5-1c y 8.5-2 juntas constituyen el controlador completo. Cuando el controlador se utiliza en conjunción con la arquitectura de la figura 8.4-1 tenemos una máquina que sumará dos números binarios.

Las señales para la máquina sumadora se dan en la figura 8.5-3. Aquí suponemos que los flip-flops de las secuencias responden a los flancos negativos de las señales de reloj. La señal para  $X$  representa la operación del interruptor. En un punto arbitrario en un ciclo de reloj se cierra el conmutador y  $X$  adopta el 1 lógico. El conmutador permanece cerrado durante un número indeterminado de ciclos de reloj y entonces se abre nuevamente en un punto arbitrario en el ciclo de reloj. Las señales mostradas son las de la salida del decodificador y también la «señal» de  $Q_2Q_1Q_0$ . La última no se genera realmente en cualquier parte del sistema y se ha incluido

para que la secuencia sea aparente cuando esté en el estado de espera. En el instante de la primera transición negativa de reloj después de haber pulsado el interruptor, el sistema deja el estado de espera (estado 0) y va al siguiente estado (estado 1), donde los estados siguientes  $Z$  y  $Z_A$  son 1. Este estado 1 persiste hasta la siguiente transición negativa del reloj después que  $X$  vuelve a  $X = 0$ . Además de las señales de habilitación  $Z$  y  $Z_A$ , todas las demás señales alcanzan el nivel de habilitación en un instante en un ciclo de reloj. Algunas señales hacen esta excursión a 1 lógico exactamente una vez cuando las secuencias circulan a través de sus estados; otras lo hacen dos veces.

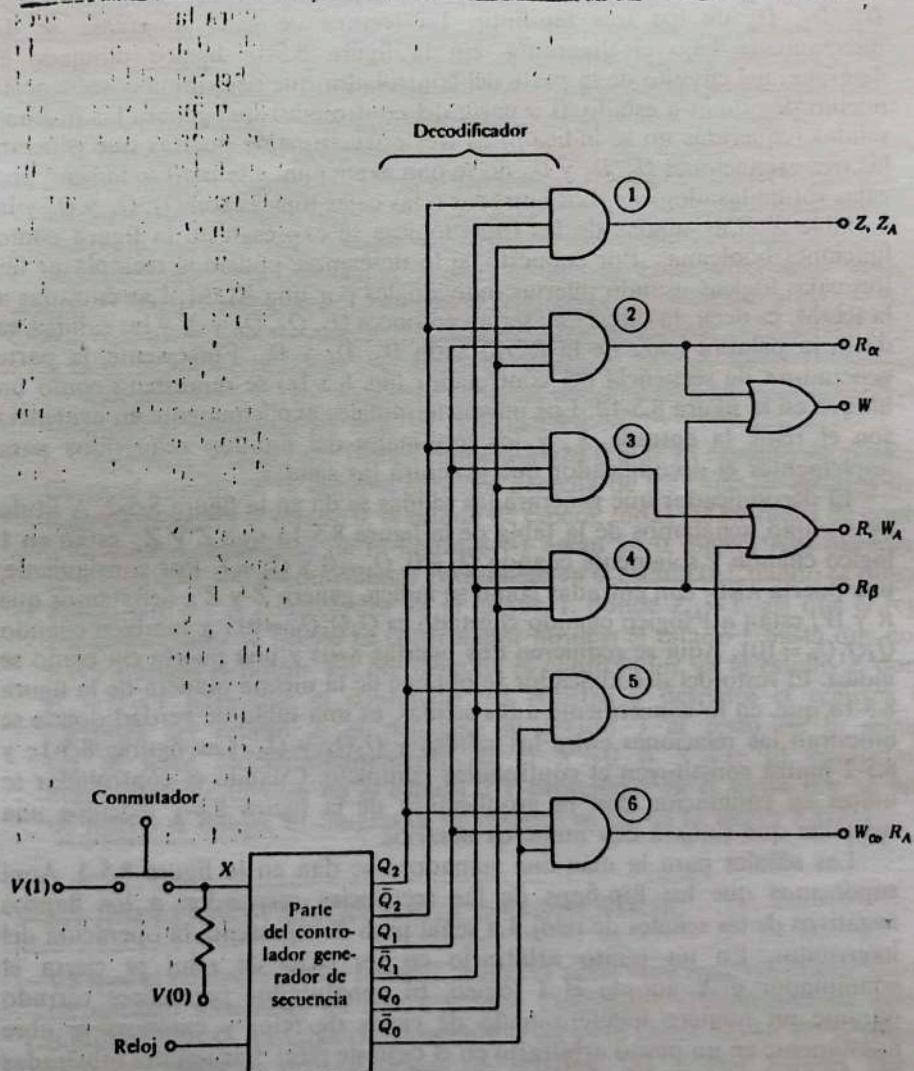


Figura 8.5-2 Decodificador utilizado con el generador de secuencia de la figura 8.5-1.

## 8.6 CONTROLADOR DE REGISTRO DE DESPLAZAMIENTO

El controlador de las secciones previas se diseñó para que necesitase un número mínimo de flip-flops. Este criterio de diseño de minimizar el número de estados con el fin de que el número de flip-flops sea mínimo se observó también en el Capítulo 7, donde discutimos detectores de secuencia. En el Capítulo 7 señalamos, sin embargo, que había un método de diseño alternativo utilizando registros de desplazamiento. El diseño con registros de desplazamiento emplea más flip-flops pero generalmente menos lógica, es decir, menos puertas para generar funciones lógicas. Y mientras, al fin, el diseño de un registro de desplazamiento no puede ser tan económico en hardware como un diseño de flip-flops mínimo, aquél tiene al menos el gran mérito de ser más ordenado y sistemático en el sentido que podemos determinar fácilmente de forma precisa lo que hace cada flip-flop. Esto generalmente no ocurre en un diseño de mínimo número de estados.

Estas consideraciones nos promueven a indagar sobre el diseño del registro de desplazamiento en el presente caso. En primer lugar señalamos que mientras el controlador diseñado en las secciones precedentes utiliza solamente tres flip-flops, éste utiliza una gran cantidad de lógica en las cajas de lógica de la figura 8.5-1 y en el decodificador de la figura 8.5-2. Segundo, notamos que las señales de la figura 8.5-3 son reminiscientes de las encontradas generalmente en los circuitos de registros de desplazamiento. Esto es, al menos en principio, bastante obvio como para utilizar un registro de desplazamiento para construir una secuencia para nuestro controlador de

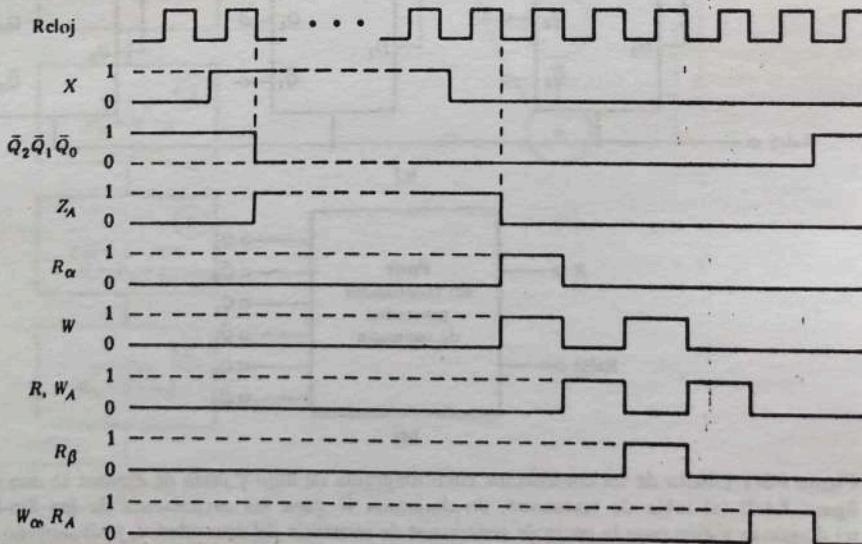


Figura 8.5-3 Señales de reloj generadas a las salidas del decodificador de la figura 8.5-2.

la máquina sumadora. Hemos de construir solamente un registro de desplazamiento de seis etapas, contador de anillo y disponemos que, en principio, el primer flip-flop esté en set y todos los otros en reset. Entonces con cada ciclo de reloj la condición set ( $Q=1$ ) progresará a lo largo del registro y en cada ciclo tendremos disponible, en sucesión a las salidas de los flip-flops, las señales habilitadoras de los ciclos de reloj que requerimos. En la práctica, en un secuenciador de registro de desplazamiento el problema de

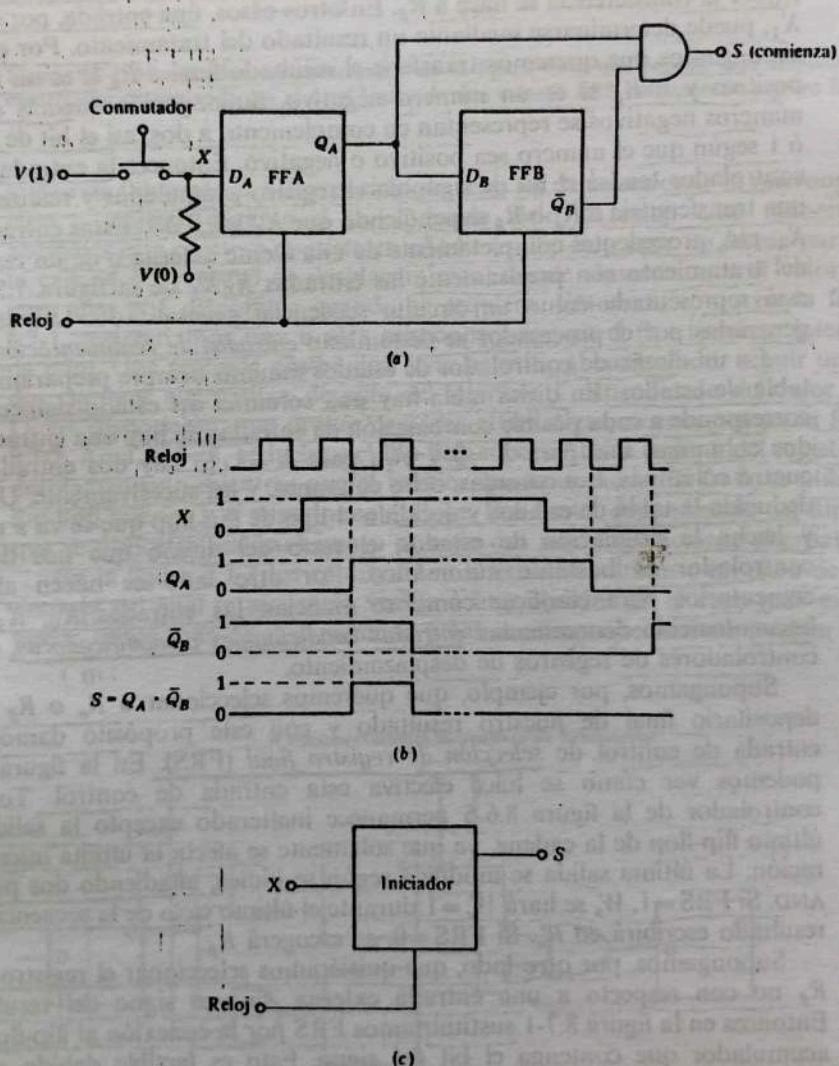


Figura 8.6-1 (a) Estructura lógica que, al cerrar el conmutador, genera una transición de  $S$  al nivel lógico 1 que persiste durante un ciclo de reloj. (b) Señales. (c) Símbolo del iniciador.

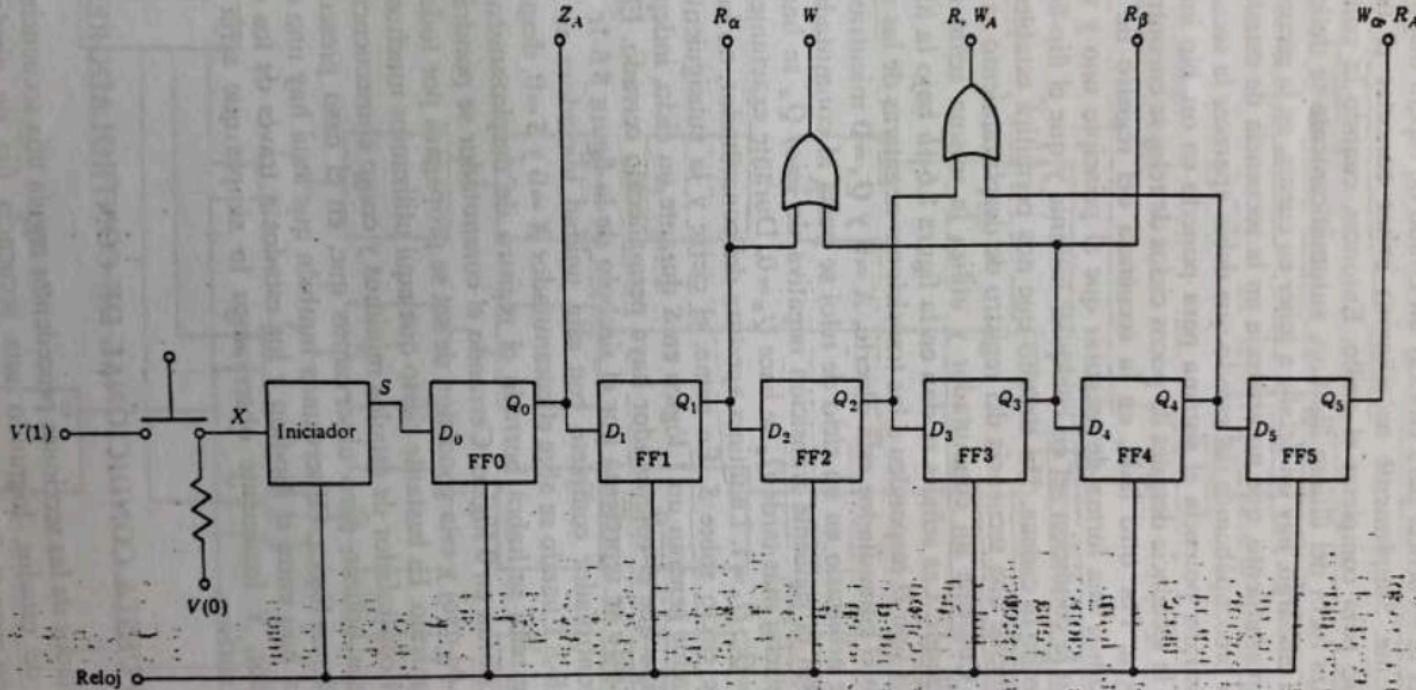


Figura 8.6-2 Controlador de registro de desplazamiento que puede servir para sustituir al controlador de estados mínimos de las figuras 8.5-1 y 8.5-2.

detener la secuencia para fabricar un estado de espera disponible se solventa fácilmente. Simplemente necesitamos evitar conectar el único flip-flop al primero para completar el anillo. Entonces, cuando la condición de set se desplaza fuera del último flip-flop, automáticamente se detiene la secuencia.

Sin embargo, hay un detalle a tener en cuenta, en la secuencia del registro de desplazamiento, que no aparecía en la secuencia de estados mínima. Allí, no nos preocupábamos del estado que debía aparecer la secuencia cuando se suministraba potencia al sistema para ponerlo en on. No importa el estado inicial, la secuencia después de pocos ciclos de reloj se encontrará en el estado de espera. Por otro lado, en la secuencia del registro de desplazamiento necesitamos una forma de asegurar que al principio uno y sólo un registro entre en la condición set en cualquier instante y que el flip-flop en set sea el primero de la cadena. Un circuito que nos permitirá establecer y comenzar propiamente una secuencia del registro de desplazamiento se da en la figura 8.6-1a. Contiene un comutador y utiliza la misma señal de reloj que el secuenciador. Las señales se dan en la figura 8.6-1b bajo la hipótesis que los flip-flops tipo D responden a las transiciones negativas de las señales de reloj. Cuando el comutador está abierto,  $X = 0$  y  $Q_A = 0$  mientras  $Q_B = 1$ . En un instante arbitrario en un ciclo de reloj se pulsa al comutador y  $X$  se hace  $X = 1$ . En la siguiente transición negativa de reloj  $Q_A$  se hace  $Q_A = 1$  y un ciclo de reloj más tarde  $Q_B$  se hace  $Q_B = 0$ . Durante exactamente un ciclo de reloj  $S = Q_A \cdot \bar{Q}_B = 1$ . La última apertura del comutador, no importa cuándo, no tiene efecto sobre  $S$ . En suma, el cierre y la subsiguiente apertura del comutador generan un 1 lógico en  $S$  durante un ciclo, independientemente del tiempo que el comutador haya permanecido cerrado. El circuito de la figura 8.6-1a se representó por el símbolo de la figura 8.6-1c.

El controlador completo con esta unidad iniciadora se muestra en la figura 8.6-2. Cuando se abre el comutador  $X = 0$  y  $S = 0$ , después de algunos ciclos de reloj se habrá borrado el registro de desplazamiento y todas las salidas estarán en 0 lógico. Cerrando el comutador se pondrá  $S = 1$  durante un ciclo de reloj y esta condición de set se propagará por todo el registro de desplazamiento. Es bastante cierto que aquí utilizamos muchos más flip-flops que en el controlador de estados mínimos y como consecuencia hay muchos estados sin utilizar, pero observamos que, en el caso presente, se emplea mucha menos lógica. Observamos también que aquí hay una correspondencia uno a uno entre el flip-flop y los estados a través de los cuales pasa el controlador. Y finalmente observamos lo simple que sería complicar el presente sistema.

## 8.7 RESPUESTA CONDICIONAL DE CONTROLADORES

El controlador de las secciones precedentes seguía una secuencia fija de ciclos a través de estados, logrando una secuencia fija de niveles lógicos de habilitación para realizar una secuencia fija de microoperaciones. En el caso más general, queremos un controlador para seguir diferentes secuencias bajo diferentes circunstancias. Para dirigir el controlador, éste tendrá disponibles

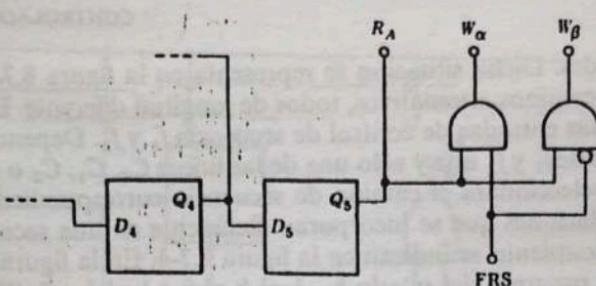
una serie de entradas  $X_0, X_1$ , etc. En algunos casos, una entrada, por ejemplo  $X_0$ , provendrá de una fuente externa al controlador y al procesador que se va a controlar, el nivel lógico de  $X_0$  se determinará por intervención humana. Por ejemplo, en nuestro sistema anterior el resultado final de la suma es, en último paso, transferido al registro  $R_a$ , pero debíamos poder escoger si el resultado se transfiere a  $R_a$  o a  $R_b$ . Y podíamos disponer entonces, por ejemplo, que cuando  $X_0 = 0$ , la transferencia se hace a  $R_a$  y cuando ponemos  $X_0 = 1$  la transferencia se hace a  $R_b$ . En otros casos, una entrada, por ejemplo  $X_1$ , puede determinarse mediante un resultado del tratamiento. Por ejemplo, supongamos que queremos transferir el resultado final a  $R_a$  si es un número positivo y a  $R_b$  si es un número negativo. Supongamos además que los números negativos se representan en complemento a dos, así el bit de signo 0 ó 1 según que el número sea positivo o negativo. Entonces la entrada  $X_1$  del controlador tendrá el bit de signo en el registro acumulador y realizaríamos una transferencia a  $R_a$  o  $R_b$  dependiendo que  $X_1$  sea 0 ó 1. Estas entradas  $X_0, X_1$ , etc., procedentes completamente de una fuente externa o de un resultado del tratamiento son precisamente las entradas  $X_0X_1$  de la figura 7.5-6 que está representada como un circuito secuencial generalizado. Las entradas generadas por el procesador se denominan entradas de realimentación.

En un diseño de controlador de estados mínimo siempre preparamos una tabla de estados. En dicha tabla hay una columna del estado-siguiente que corresponde a cada posible combinación de entradas. Si hay una entrada, hay dos columnas, una para  $X=0$  y otra para  $X=1$ . Si hay dos entradas hay cuatro columnas, tres entradas, ocho columnas, y así sucesivamente. Una vez deducida la tabla de estados y decidido el tipo de flip-flop que se va a utilizar y hecha la asignación de estados, el resto del diseño que nos lleva al controlador es bastante automático. Por otro lado se hacen algunos comentarios para clarificar cómo se manejan las entradas  $X_0, X_1$ , etc., frecuentemente denominadas entradas condicionales o modificadoras, en los controladores de registros de desplazamiento.

Supongamos, por ejemplo, que queremos seleccionar a  $R_a$  o  $R_b$  como depositario final de nuestro resultado y con este propósito damos una entrada de control de selección de registro final (FRS). En la figura 8.7-1 podemos ver cómo se hace efectiva esta entrada de control. Todo el controlador de la figura 8.6-5 permanece inalterado excepto la salida del último flip-flop de la cadena, ya que solamente se afecta la última microoperación. La última salida se modifica según se indica, añadiendo dos puertas AND. Si  $FRS = 1$ ,  $W_a$  se hará  $W_a = 1$  durante el último ciclo de la secuencia y el resultado escribirá en  $R_a$ . Si  $FRS = 0$ , se escogerá  $R_b$ .

Supongamos, por otro lado, que quisieramos seleccionar el registro  $R_a$  o  $R_b$  no con respecto a una entrada externa, sino al signo del resultado. Entonces en la figura 8.7-1 sustituiríamos FRS por la conexión al flip-flop del acumulador que contenga el bit del signo. Esto es factible debido a que durante el intervalo en que se haga la transferencia final a  $R_a$  o  $R_b$  no se ha borrado todavía el acumulador y el bit de signo está disponible.

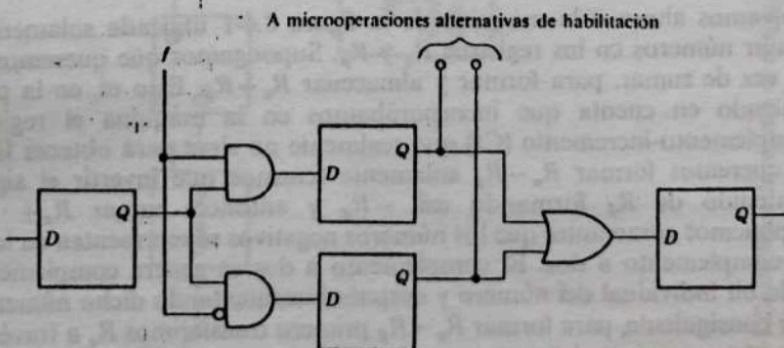
Consideremos a continuación un caso en el que haya una entrada de realimentación f al controlador y supongamos que se necesita que el valor



**Figura 8.7-1** Modificador del controlador de la figura 8.6-2 que permite una respuesta del controlador determinada por el nivel lógico de una entrada FRS (selección de registro final).

lógico de  $f$  durante el  $k$ -ésimo ciclo de reloj ha de seleccionar la microoperación a realizar durante el  $(k+1)$ -ésimo ciclo de reloj. Una manera de acometer este objetivo consiste en añadir un flip-flop (que no forme parte de la cadena del registro de desplazamiento) donde almacenamos el valor de  $f$  para que esté disponible cuando se requiera. Entonces, como en la figura 8.7-1, realizariamos una u otra microoperación en el  $(k+1)$ -ésimo intervalo de reloj. Una segunda posibilidad sería permitir al controlador seguir uno de dos caminos alternativos después del intervalo  $k$  dependiendo del valor de  $f$ . Dicha disposición, que provee caminos alternativos, se muestra en la figura 8.7-2. Aquí, si  $f=1$ , se incluye el flip-flop superior en la cadena del registro de desplazamiento mientras que el inferior queda excluido. Si  $f=0$ , se utiliza el flip-flop inferior y el superior se excluye. Por tanto, dependiendo del flip-flop que se utilice se realizará una u otra de dos microoperaciones alternativas.

En la figura 8.7-2 los caminos son alternativos durante un ciclo de reloj después del cual la secuencia vuelve a uno común. En el caso más general puede haber más de dos caminos y los caminos alternativos pueden tener



**Figura 8.7-2** Controlador del registro de desplazamiento que permite al nivel lógico de la entrada de realimentación  $f$  establecida durante un intervalo para controlar la microoperación realizada durante el siguiente intervalo.

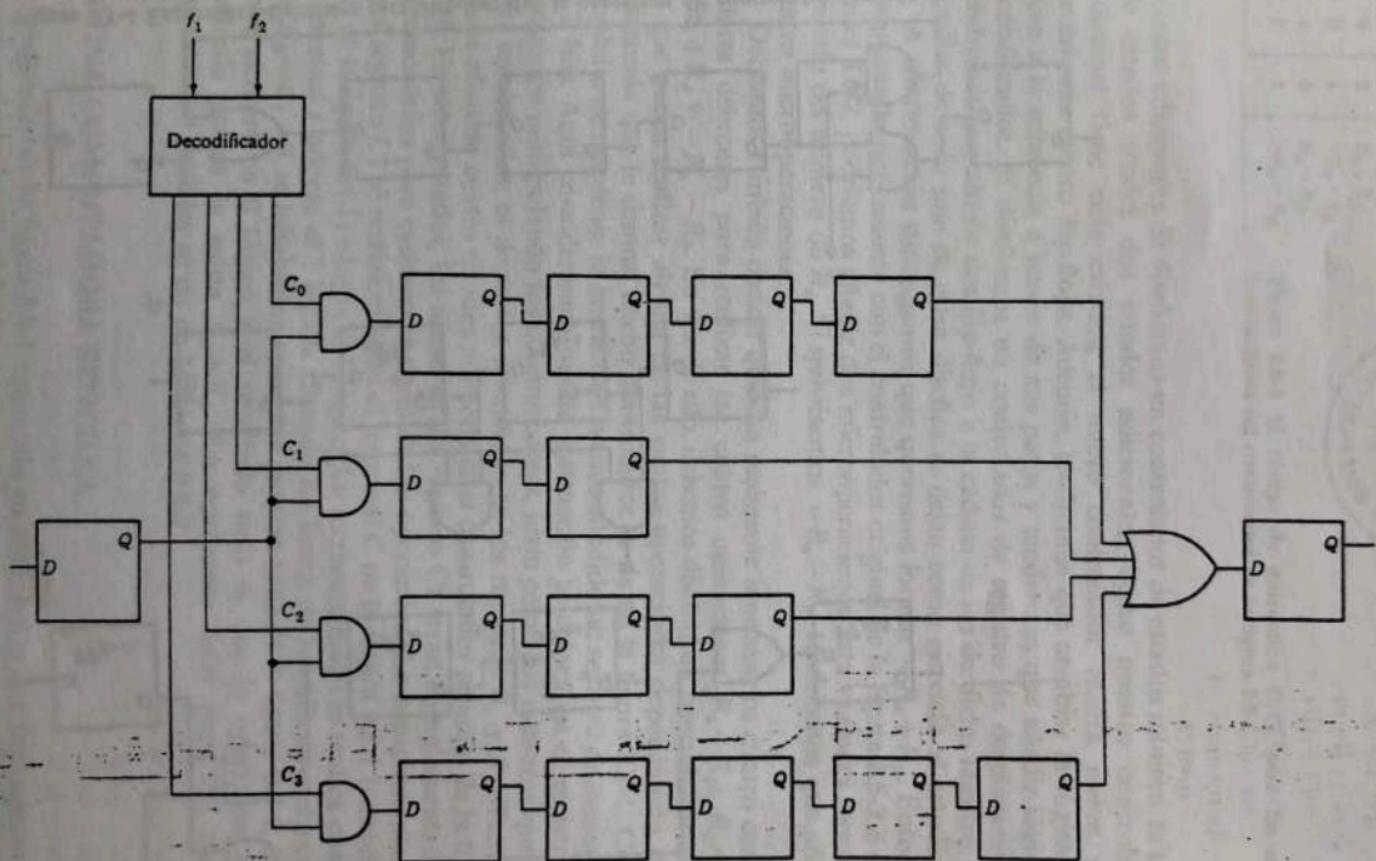


Figura 8.7-3 Controlador que permite cuatro secuencias diferentes de longitudes variables dependiendo de las entradas  $f_1$  y  $f_2$ .

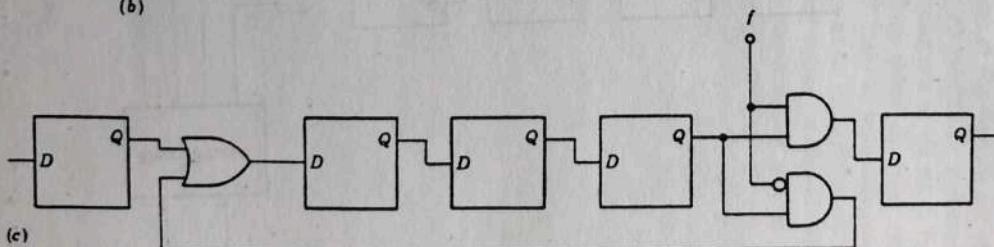
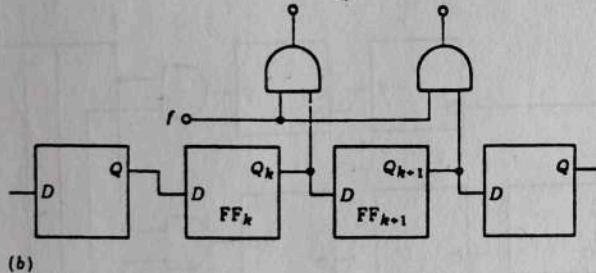
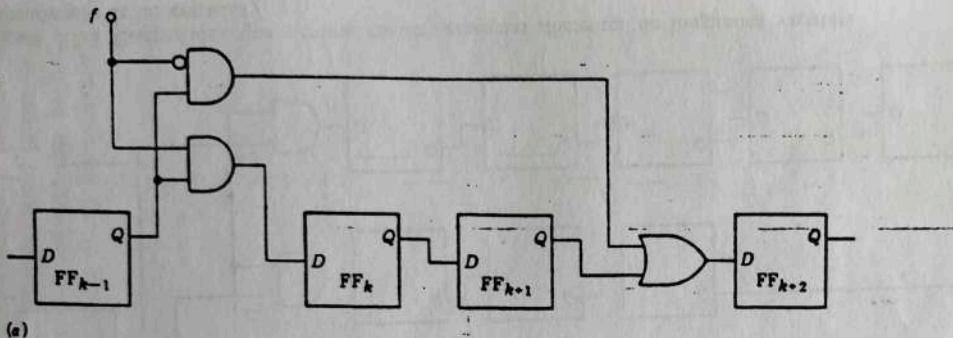


Figura 8.7-4 Esquemas adicionales para disponer que la operación del controlador dependa de los niveles lógicos de una entrada modificable  $f$ . (a) Cuando  $f=0$ , algunos estados se saltan. (b) Cuando  $f=0$ , algunos estados se vuelven ineffectivos. (c) Cuando  $f=0$ , algunos estados se repiten.

diferentes longitudes. Dicha situación se representa en la figura 8.7-3, donde se indican cuatro caminos alternativos, todos de longitud diferente. El camino se selecciona por las entradas de control de secuencia  $f_1$  y  $f_2$ . Dependiendo de los valores lógicos de  $f_1$  y  $f_2$  una y sólo una de las líneas  $C_0$ ,  $C_1$ ,  $C_2$  o  $C_3$  estará en 1 lógico y se seleccionará el camino de secuencia correspondiente.

Otras manipulaciones que se incorporan fácilmente en una secuencia del registro de desplazamiento se indican en la figura 8.7-4. En la figura 8.7-4a si  $f=1$ , la secuencia progresiva del estado  $k-1$  al  $k$  al  $k+1$  al  $k+2$ . (Definimos aquí el estado  $k$  como el estado donde el flip-flop  $k$  se encuentra en el estado de set.) Sin embargo, cuando  $f=0$  los estados  $k$  y  $k+1$  son eliminados y avanzamos del estado  $k-1$  al  $k+2$ . En la figura 8.7-4b cuando  $f=1$  los estados  $k$  y  $k+1$  no son eliminados pero se vuelven ineficaces. Las microoperaciones que podían haberse realizado cuando  $Q_k$ , y más tarde  $Q_{k+1}$  estaban a 1 lógico no se realizarán. En la figura 8.7-4a y b el resultado final es el mismo que si las microoperaciones a realizar en los estados  $k$  y  $k+1$  realmente no se realizasen. El esquema de la figura 8.7-4a tiene el mérito de ahorrar tiempo, ya que la secuencia elimina los estados  $k$  y  $k+1$ . Sin embargo, si como no parece infrecuente la secuencia debe esperar entonces para que alguna operación se realice en otra parte o se complete, los esquemas de la figura 8.7-4a y b son comparables. Finalmente, necesita decirse duramente que el número de estados de salto o microoperaciones es completamente ajustable.

La figura 8.7-4c indica cómo podemos actuar para hacer que una secuencia vuelva a repetirse, parcial o totalmente (la repetición se realiza tantas veces como queramos). Si  $f=1$  la secuencia continúa su curso normal. Si  $f=0$ , la secuencia repite una serie de estados y continúa con este patrón de repetición hasta que nuevamente pongamos  $f$  en  $f=1$ .

## 8.8 SECUENCIA PARA LA RESTA

Volvamos ahora a la máquina de la figura 8.4-1 utilizada solamente para sumar números en los registros  $R_a$  y  $R_b$ . Supongamos que queremos restar, en vez de sumar, para formar y almacenar  $R_a - R_b$ . Esto es, en la práctica, teniendo en cuenta que incorporábamos en la máquina el registro de complemento-incremento (CI) que realmente no sirve para obtener la suma. Si queremos formar  $R_a - R_b$  solamente tenemos que invertir el signo del contenido de  $R_b$  formando así  $-R_b$  y entonces sumar  $R_a + (-R_b)$ . Suponemos como antes que los números negativos se representan en la forma de complemento a dos. El complemento a dos se genera complementando cada bit individual del número y después incrementando dicho número en 1. Por consiguiente, para formar  $R_a - R_b$  primero transferimos  $R_a$  a través de CI al registro acumulador como antes, entonces transferimos  $R_b$  a CI, donde primero complementamos y después incrementamos antes de hacer la transferencia de CI al acumulador.

Para realizar la sustracción necesitamos modificar el controlador para que suministre dos microoperaciones adicionales que correspondan a dos

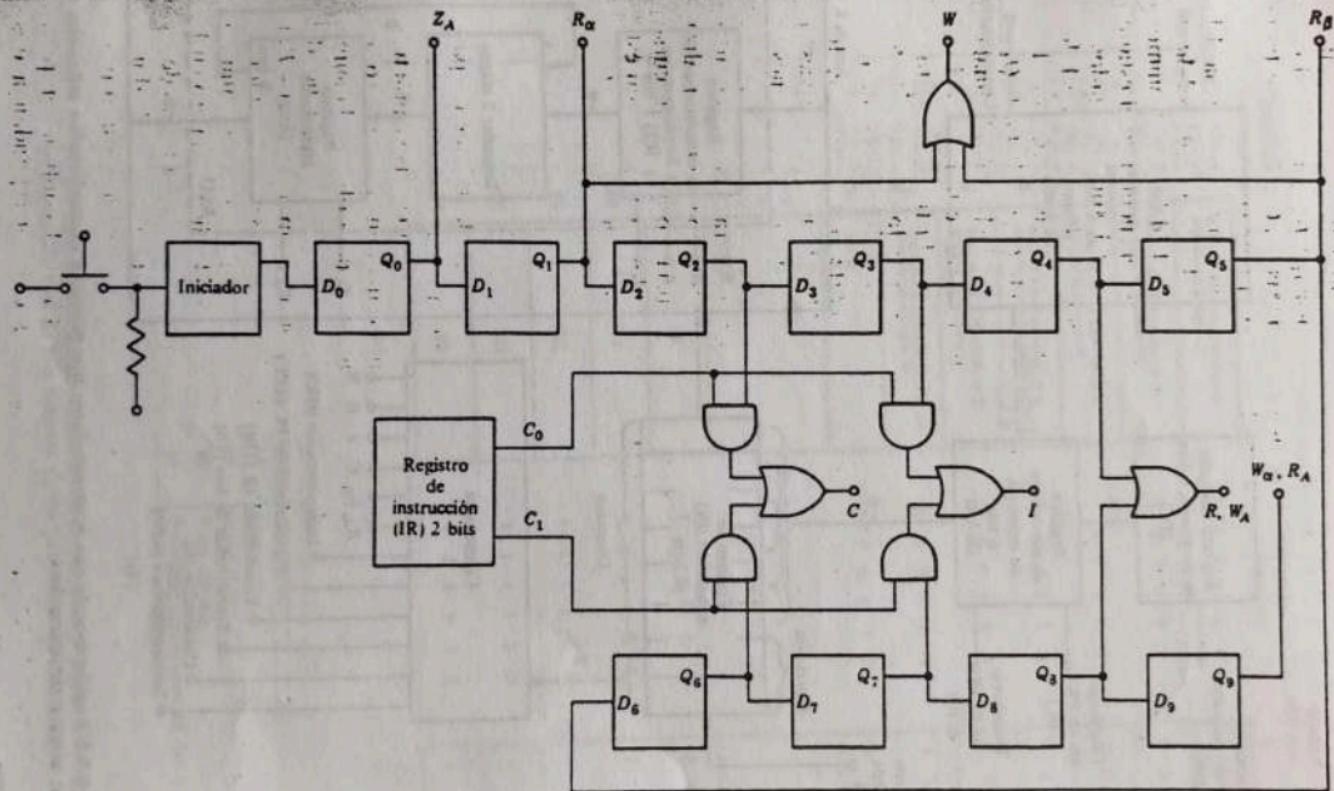


Figura 8.8-1 Controlador de registro de desplazamiento utilizado con la arquitectura de la figura 8.4-1. El controlador colocará en el registro  $R_x$  una de las cuatro cantidades  $\pm R_x$  y  $\pm R_B$ , dependiendo de la instrucción del registro de instrucciones.

Instrucción		Operación total
$C_1$	$C_0$	
0	0	$R_\alpha + R_\beta$
0	1	$-R_\alpha + R_\beta$
1	0	$R_\alpha - R_\beta$
1	1	$-R_\alpha - R_\beta$



Figura 8.8-2 El código de instrucción  $C_1C_0$  para las cuatro instrucciones del controlador de la figura 8.8-1.

estados adicionales. Si diseñamos un controlador de estados mínimo, la tabla de estados tendrá dos estados adicionales; como nuestro controlador adicional tiene siete estados, el nuevo controlador tendrá nueve, y se necesitarán cuatro flip-flops. Además, tendremos que cambiar la lógica que lleva a la secuencia a través de sus pasos y tendremos que añadir lógica al decodificador. Si diseñamos un controlador de registro de desplazamiento tendremos que añadir dos flip-flops a la cadena de los de dicho registro. Los detalles de cada uno de estos diseños se dejan como ejercicio al estudiante.

A continuación supongamos que queremos formar  $-R_\alpha + R_\beta$ . Entonces, comenzando nuevamente con el controlador original de la figura 8.5-2, o de la figura 8.6-2, podríamos añadir dos microoperaciones, esta vez para formar el opuesto del número de  $R_\alpha$ . Si quisiésemos  $-R_\alpha - R_\beta$ , tendríamos que añadir cuatro microoperaciones.

Debíamos también decidir que son realmente innecesarios cuatro controladores diferentes para realizar las cuatro operaciones  $R_\alpha + R_\beta$ ,  $R_\alpha - R_\beta$ ,  $-R_\alpha + R_\beta$  y  $-R_\alpha - R_\beta$ . En vez de ello, debemos diseñar un controlador sencillo que pueda realizar alguna de las cuatro operaciones dependiendo de la instrucción que le demos. Dicho controlador se da en la figura 8.8-1. Con el propósito de mantener la instrucción hemos añadido un registro de instrucción de 2 bits. Aquí consideraremos que la instrucción se coloca en el registro de instrucción manipulando los commutadores, justo como los números que van a ser combinados; es decir, los operandos en los registros  $R_\alpha$  y  $R_\beta$ . También hemos añadido cuatro estados adicionales al controlador original de la figura 8.6-2. En estos estados, los números del registro CI serán complementados e incrementados para cambiar el signo de los operandos. La complementación en el registro CI se realiza cuando el terminal C de la figura 8.8-1 (conectado a C en la figura 8.4-1) adopta el 1 lógico. La incrementación se realiza cuando el terminal I adopta el 1 lógico. Cuando las líneas de los bits de instrucción son  $C_0 = C_1 = 0$ , se inhabilitan todas las puestas AND de la figura 8.8-1, C e I están siempre en 0 lógico y el resultado neto es que la operación total realizada genera la suma  $R_\alpha + R_\beta$ . La operación total para las cuatro instrucciones posibles se da en la figura 8.8-2.

## 8.9 UNA COMPUTADORA SENCILLA

El controlador de la figura 8.8-1 operando en conjunción con los registros de almacenamiento y controlables organizados en la arquitectura de la figura 8.4-1

nos permitirá combinar aritméticamente dos números mediante suma y resta. Para utilizar la máquina pondremos los números en  $R_x$  y  $R_B$ , presumiblemente a mano mediante conmutadores, ya que no hemos hecho otras previsiones. Entonces podríamos poner manualmente una instrucción en el registro de instrucciones (IR). Después de eso, todo se realiza automáticamente. Solamente necesitamos pulsar el botón de comenzar, y después de un rato encontraremos nuestro resultado en  $R_x$ .

Supongamos ahora que queremos hacer nuestra máquina más elaborada. Primero queremos poder tratar con más de dos números u operandos. Así podemos realizar una suma de esos operandos pero seleccionándolos entre un gran número. O, disponiendo un gran número de operandos, podemos combinar muchos. Una modificación obvia que se ocurre entonces es reemplazar los registros  $R_x$  y  $R_B$  por un gran array de registros. Dicho array es, por supuesto, una memoria como la descrita en el Capítulo 6. Si queremos cambiar fácilmente los operandos necesitaremos una RAM.

En el controlador de la figura 8.8-1 una serie de pasos de secuencia tratan a los operandos almacenados en  $R_x$  y  $R_B$ . Es claro que este modelo nos conduciría a una secuencia muy grande si hubiese muchos operandos. Cada nuevo operando añade pasos a la secuencia. Sin embargo, reconocemos que cada operando está realmente sujeto a las mismas operaciones. El operando se transfiere desde un registro de almacenamiento (o mejor desde una posición de memoria) al registro CI y de ahí a través del sumador al acumulador. Si necesitamos cambiar el signo del operando, complementamos e incrementamos en el registro CI. En otra situación, este registro no hace nada. Esta secuencia de microoperaciones se repite de nuevo para cada operando. Por tanto, parece que realmente nos podemos arreglar con un controlador que tenga una secuencia corta que trate exactamente un operando. Sin embargo, con dicho controlador de secuencia necesitaríamos algún mecanismo para ajustar cada nuevo operando que se vaya a tratar. Una forma de ajustar la instrucción es, realmente, detener la secuencia de control después de tratar cada operando y entonces manualmente cambiar la instrucción antes de permitir a la secuencia tratar al siguiente operando. Pero ya que tenemos memoria, podemos almacenar en ella la información relativa a como se trata cada operando y la operación completa puede hacerse automáticamente. Con estas consideraciones en cuenta volvamos ahora a la figura 8.9-1, que visualiza un sistema que (con alguna intervención manual) nos permitirá combinar aritméticamente un gran número de operandos automáticamente.

El sistema tiene una memoria RAM. Para ser específicos, supongamos una memoria de 64 palabras, cada una de 8 bits. Una posición de memoria se direcciona con 6 bits ( $2^6 = 64$ ). La memoria tiene una entrada de *habilitación* y otra de *lectura/escritura*. Cuando la *habilitación* = 1, la memoria se conecta al bus (de 8 bits) y cuando la *habilitación* = 0, el bus se aisla de la memoria. Cuando la *habilitación* es = 1, la memoria leerá una palabra en el bus o escribirá una palabra en memoria, dependiendo de que *lectura/escritura* sea 1 ó 0. La posición de memoria de la que se lee o en la que se escribe una palabra

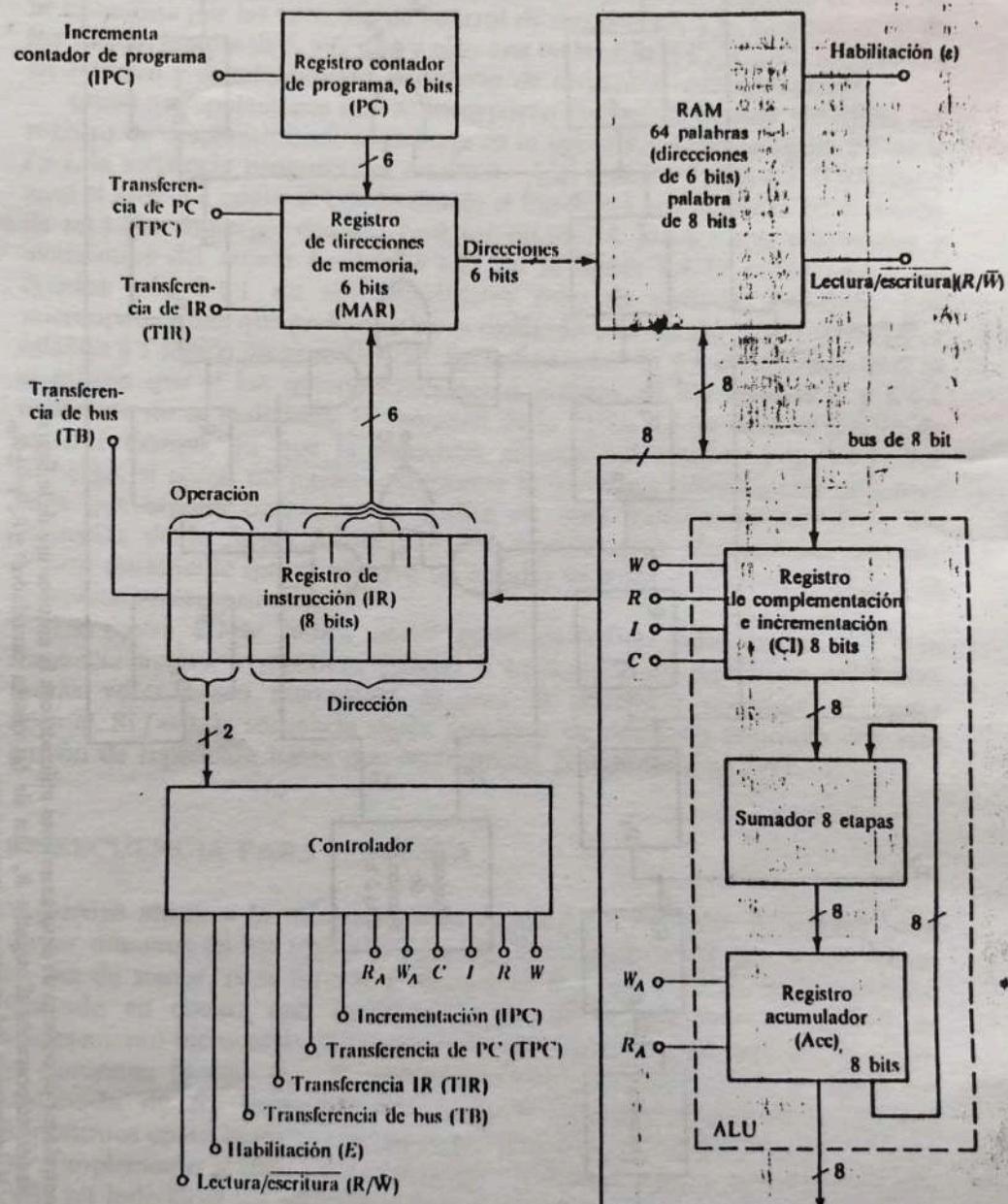


Figura 8.9-1 Arquitectura de una «computadora» que permitirá la combinación aritmética de un gran número de operandos.

Dirección de memoria		Dirección de memoria	Resta del Acc	Dirección de memoria 59
0	Resta del Acc el contenido de la dirección de memoria 59	0 0 0 0 0 0	1 0 x 1 1 1 0 1 1	
1	Suma al Acc el contenido de la dirección de memoria 60	0 0 0 0 0 1	0 1 1 1 1 1 0 0	
2	Resta del Acc el contenido de la dirección de memoria 61	0 0 0 0 1 0	1 0 1 1 1 1 0 1	
3	Suma al Acc el contenido de la dirección de memoria 62	0 0 0 0 1 1	0 1 1 1 1 1 1 0	
4	Suma al Acc el contenido de la dirección de memoria 63	0 0 0 1 0 0	0 1 1 1 1 1 1 1	
5	Transfiere el contenido del Acc a la dirección de memoria 59	0 0 0 1 0 1	1 1 1 0 0 1 1 1	
6	Alto	0 0 0 1 1 0	0 0 x x x x x x	
•		•	•	
59	49	1 1 1 0 1 1	0 0 1 1 0 0 0 1	
60	-79	1 1 1 1 0 0	1 0 1 1 0 0 1 1	
61	-52	1 1 1 1 0 1	1 1 0 0 1 1 0 0	
62	121	1 1 1 1 1 0	0 1 1 1 1 0 0 1	
63	82	1 1 1 1 1 1	0 1 0 1 0 0 1 0	

(a)

(c)

Código	Instrucción
00	Alto
01	Suma al Acc
10	Resta del Acc
11	Transfiere el contenido del Acc a

(b)

Figura 8.9-2 (a) Posible contenido de la memoria de la figura 8.9-1. (b) Código de instrucción. (c) Contenido real de bits binarios de las posiciones de memoria.

se determina por los seis bits de dirección. La doble cabeza de flecha que conecta la memoria al bus indica que la transferencia de información entre el bus y memoria es bidireccional.

En la figura 8.9-2 ilustramos lo que tipicamente debía estar contenido en la memoria de nuestra máquina sencilla. En la figura 8.9-2a hemos escrito en palabras y números decimales el contenido de algunas posiciones de memoria. En las posiciones 0 a 6 hemos escrito un *programa* de instrucciones. En el otro extremo de la memoria hemos almacenado algunos operandos. Si la máquina realizase las instrucciones dadas en el orden listado, entonces calcularía la cantidad  $-(49) + (-79) - (-52) + (121) + (82) = +127$ , como puede verse observando los contenidos de las posiciones de memoria 59 a 63. La máquina transferirá esta suma acumulada del registro acumulador a la posición 39 de memoria y entonces se parará y esperará a una intervención humana. Como veremos, hay un propósito al colocar las instrucciones en posiciones sucesivas de memoria. No es importante que las instrucciones y los operandos estén colocados en los extremos opuestos de la memoria. Lo único que se requiere es que las instrucciones se coloquen (en posiciones consecutivas) en una parte de la memoria y los operandos en otra parte. El orden indicando en la figura 8.9-2 se sugiere principalmente con el fin de realizar los cálculos ordenadamente. Las posiciones de memoria no indicadas en la figura tienen algún contenido. (Una posición de memoria borrada con contenido 00...00 tiene un contenido sin nada.) Pero el contenido de estas posiciones de memoria no es relevante en nuestra discusión actual. Nos proponemos almacenar nuestro resultado en la posición 39 (no mostrada). El contenido de la posición 39 no se conoce y es irrelevante. Cuando la instrucción se ejecuta para que escriba el resultado en la posición 39, el contenido previo de esa posición se perderá.

Señalamos que hemos utilizado cuatro instrucciones: suma, resta, transferencia y parada. Arbitrariamente, utilizamos el código de 2 bits de la figura 8.9-2b para representar esas instrucciones. En la figura 8.9-2c hemos vuelto a escribir la memoria en forma binaria. En las posiciones que contienen operandos hemos reemplazado sencillamente los números decimales por binarios. En las posiciones de memoria que contienen instrucciones hemos dispuesto arbitrariamente que los dos bits de más a la izquierda representen la instrucción y los 6 bits restantes especifiquen la posición que contiene el operando. Esta asignación de significado a los bits se indica expresamente para la primera posición de memoria. Las posiciones de memoria 0 a 5 contienen instrucciones que realizan una operación y se refieren a una posición específica de memoria donde se almacena el operando. La posición de memoria 6 realiza una operación pero como no hay operando, es irrelevante la dirección de seis bits.

Observemos que las palabras de memoria tienen una longitud de 8 bits. Si nos proponemos representar números negativos en la forma del complemento a dos, el rango de los números puede acomodarse desde +127 a -128. Entonces debemos ser cuidadosos de asegurarnos cómo acumula nuestra máquina los números que estamos combinando; nunca se requerirá que la suma acumulada sobrepase el rango permitido. Los números introducidos en

las posiciones de memoria 59 a 63 se han seleccionado arbitrariamente excepto que hemos observado esta ligadura con respecto al rango.

Volviendo a la figura 8.9-1 señalamos que la sección en la caja a trazos es el sistema de la figura 8.4-1 son los registros  $R_a$  y  $R_b$ . (Estos registros se han sustituido por la parte de memoria que almacena los operandos.) Este pequeño sistema realiza aritmética (suma e incrementa) y lógica (complementación) y razonablemente se denomina unidad aritmética lógica (ALU), aunque difiere en muchos aspectos de la ALU de la figura 5.13-1. Por simplicidad hemos dejado fuera el terminal de borrador  $Z_A$ , ya que el acumulador puede borrarse empleando las instrucciones de la figura 8.9-2b (Problema 8.9-1).

Continuando nuestro examen de la figura 8.9-1 señalamos la presencia de un registro contador de programa (PC), registro de instrucción (IR), registro de direcciones de memoria (MAR) y, finalmente, el controlador que pone a nuestra pequeña computadora sobre sus pasos y que ya hemos diseñado. El propósito detallado de cada registro será discutido brevemente. De momento notemos simplemente los caminos disponibles para las transferencias de palabras (o partes de palabras) y las facultades que han sido incorporadas a cada registro. Señalamos que hay un bus de 8 bits al que la memoria tiene una conexión bidireccional. La ALU puede aceptar una palabra del bus sobre su lado de entrada y suministrar una palabra al bus de su lado de salida. El registro de instrucción puede aceptar una palabra del bus. El registro de direcciones de memoria tiene dos entradas de control que pueden utilizarse para transferir al MAR la palabra de 6 bits del contador de programa o los 6 bits de más a la derecha del registro de instrucción. Los dos bits de más a la izquierda del registro de instrucciones se hacen disponibles al controlador (no transferidos). Por consiguiente, esta conexión de dos bits se indica a trazos en vez de por una línea de salida. La única operación que puede realizar el contador de programa es la de incrementar. Finalmente, el controlador tiene una línea de salida de control correspondiente a cada línea de entrada de control de cada registro y de la memoria. Cualquier microoperación se realiza cuando la línea de control correspondiente adopte el nivel de habilitación. Con la excepción de la memoria, todos los registros y el controlador tienen reloj, y el momento actual en el que un registro acepta una transferencia y se incrementa en el instante de la transición de disparo de reloj. La línea de la señal de reloj que se distribuye a través de todo el sistema de la figura 8.9-1 no está indicada en el dibujo.

## 8.10 OPERACIÓN DE LA COMPUTADORA

Para ver cómo opera nuestra computadora consideremos que nuestra memoria se carga como en la figura 8.9-2c. Podemos imaginar que para efectuar esta carga estaba desconectada temporalmente del sistema y que las entradas (direcciones, datos, *habilitación*, *lectura/escritura*) se aplicaban manualmente. También suponemos que en principio el controlador de programa (PC) y el registro acumulador están borrados. (No debe importar si los

otros registros están borrados inicialmente.) Ahora listaremos en las tablas 8.10-1 y 8.10-2 cada ciclo de reloj, la secuencia de microoperaciones a través de las cuales el controlador debe pasar al sistema para que éste tome nota de la primera instrucción, realice su intento y se prepare para la siguiente instrucción. Cuando es factible realiza más de una microoperación durante el transcurso de un ciclo de reloj, por ello tomaremos nota de esta característica con el fin de ahorrar tiempo. Especialmente se señala que las únicas operaciones especificadas en la tabulación son aquellas que estarán afectadas por el hecho de que una o más salidas de control del controlador van al nivel de habilitación.

Tabla 8.10-1 Ciclo de búsqueda

Ciclo de reloj	Descripción simbólica de la operación	Línea de control a habilitar
1. Transfiere el contenido del contador de programa al registro de direcciones de memoria	$PC \rightarrow MAR$	TPC
2. Transfiere la instrucción direccionada (en la posición 000000) al registro de instrucción mediante: 1) habilitación de memoria al conectarla al bus; 2) poniendo $R/W$ a 1 al leer memoria, y 3) transfiriendo la palabra del bus al registro de instrucción, incrementando el contador de programa preparándolo para llamar la siguiente instrucción cuando se ha completado la respuesta a la primera	$M \rightarrow IR$ («M» representa palabra de memoria direccionada)	$E_1, R/W, TB$
	$PC + 1 \rightarrow PC$	IPC

En las operaciones listadas en la tabla 8.10-1 hemos trasladado las primeras instrucciones de memoria al registro de instrucciones. Esta parte del ciclo de operaciones de la máquina se denomina ciclo de búsqueda. Ahora que la primera instrucción está disponible, la máquina procederá a responder a las instrucciones. El ciclo de operaciones por el que se realiza esta respuesta se denomina ciclo de ejecución. Estas operaciones del ciclo de búsqueda, por supuesto, se realizan bajo la supervisión del controlador, pero la operación de este durante el ciclo de búsqueda es independiente de los bits de más a la izquierda en el registro de instrucción, disponibles para el controlador. En la práctica, como no borrábamos el registro de instrucción, no sabíamos cuáles eran esos bits. Sin embargo, ahora que hemos buscado y almacenado esta primera instrucción en el registro de instrucción, el tratamiento de este punto seguirá un curso dependiente de la instrucción, convenido por los dos bits de operación de la misma. Para la instrucción llamada sustracción, su ejecución se realiza como muestra la tabla 8.10-2.

Tabla 8.10-2 Ciclo de ejecución

Ciclo de reloj	Descripción simbólica de la operación	Línea de control a habilitar
3. Transfiere la parte dirección del registro (6 bits de la derecha) al registro de direcciones de memoria (dirección 59)	IR (ADD)→MAR	TIR
4. Transfiere la palabra direccionada de memoria al bus y de ahí al registro CI	M→BUS BUS→CI	E, R/I; W
5. Complementa CI	CI→CI	C
6. Incrementa CI	CI+1→CI	I
7. Salida del registro sumador al registro acumulador	Sumador→Acc	WA

La máquina ha realizado ahora las primeras instrucciones. Dispondremos ahora que, consiguientemente, el controlador que diseñemos haya completado su secuencia completa y vuelto a su comienzo. Por tanto, la siguiente operación a realizar es transferir, de nuevo, el contador de programa al registro de direcciones de memoria. Pero recalcamos que hemos incrementando el contador de programa. Como consecuencia, la instrucción buscada será la segunda (en la posición 00001). La segunda instrucción se ejecutará como la primera, excepto que como se trata de una suma en lugar de una resta se eliminan las operaciones de complementación e incrementación. Así vemos como actúan las secuencias del controlador: busca después ejecuta, busca después ejecuta, etc. La operación de búsqueda es siempre la misma; las operaciones durante la ejecución dependen, por supuesto, de la instrucción.

Con respecto al sencillo sistema de la figura 8.9-1 tomamos una serie de decisiones (algunas bastante arbitrariamente, otras basadas en la experiencia.) Estas decisiones se refieren al número y función de los registros, a los tipos de interconexión entre sí y entre ellos y memoria, al número de palabras de memoria, al número de bits por palabra, al número y tipo de operaciones que la ALU puede realizar, etc. Estas materias constituyen la arquitectura y organización de la computadora. Una vez establecida una arquitectura y organización, queda la tarea de diseño del controlador. (El controlador para nuestra máquina se diseña en la siguiente sección.)

Hay por supuesto muchas arquitecturas y organizaciones que pueden asumirse por una parte de la máquina computadora. Después de muchos años de experimentación con una amplia gama de posibilidades, las máquinas computadoras de la actualidad incorporan generalmente una serie de características comunes, algunas de las cuales se han visto en nuestra computadora sencilla y que apuntamos ahora.

Primero señalamos que la máquina tiene memoria, en la que en principio almacenamos todas las instrucciones necesarias en la realización de un cálculo. Por esta razón, la máquina se denomina computadora de programa almacenado. Como la máquina está completamente instruida, no necesitare-

mos interrumpirla para darle direcciones adicionales para sus cálculos. La memoria almacena no sólo las instrucciones, sino también los operandos y los resultados de los cálculos. Por tanto, tendremos que hacer frecuentes referencias a memoria para leer de ella y escribir en ella. La posición de la dirección se encuentra en el registro de *direcciones de memoria (memory address register, MAR)* y no es sorprendente que se acceda al MAR desde distintas direcciones. En nuestro caso podemos acceder al MAR desde el contador de programa y desde el registro de instrucciones. (Computadoras más sofisticadas están provistas de medios adicionales indirectos y directos de acceso al MAR.)

A continuación, señalamos que nuestra máquina tiene una unidad aritmético-lógica en la que se realizan todas las operaciones lógicas y aritméticas. El resto de la máquina (además del controlador) no consta de más de un array de registros. Y los registros de almacenamiento no hacen nada. Exactamente constituyen el «papel de escribir» en el que escribimos las palabras digitales que necesitemos encontrar en futuras referencias. Nuestra ALU también es bastante simple; complementa, incrementa y suma. ALU más elaboradas realizan estas funciones y también operaciones lógicas (AND, OR, etc.), desplazamientos a izquierda o derecha, etc.

Observemos que al manipular la máquina se dispone que una serie de materias «se comprendan». Cuando se ha realizado una instrucción, se «comprende» que la siguiente instrucción esté en la siguiente posición de memoria. Por tanto, no se necesita especificar la posición de dicha instrucción. Nuevamente, cuando se va a realizar una suma la instrucción especifica solamente uno de los operandos de la suma. Se «comprende» que el otro esté en el registro acumulador. Finalmente, la instrucción no da indicación donde debe almacenarse el resultado de la suma. Se «comprende» que el resultado pueda dejarse en el acumulador. Todas estas compresiones conllevan una economía considerable en la longitud de palabra. Gracias a estas compresiones que hemos incorporado en la máquina, en una instrucción solamente se necesita especificar una operación y una dirección de operando. Sin estas compresiones en una instrucción se tendría que indicar la operación, la fuente del primer operando, la del segundo, el sitio donde debe almacenarse el resultado y la fuente de la instrucción siguiente. Por supuesto, esta instrucción así elaborada requerirá muchos más bits que las instrucciones más simples posibles debida a las compresiones.

## 8.11 DISEÑO DEL CONTROLADOR DE LA COMPUTADORA

Un diseño para el controlador requerido de la figura 8.9-1 se da en la figura 8.11-1. En este diseño, con el fin de conseguir simplicidad visualizaremos un deliberado y descuidado extremo para economizar hardware. Como es habitual, el elemento de comienzo y todos los flip-flops son gobernados por una señal de reloj común (no mostrada explícitamente). Al cerrar el conmutador comienza la secuencia del registro de desplazamiento. Cuando la

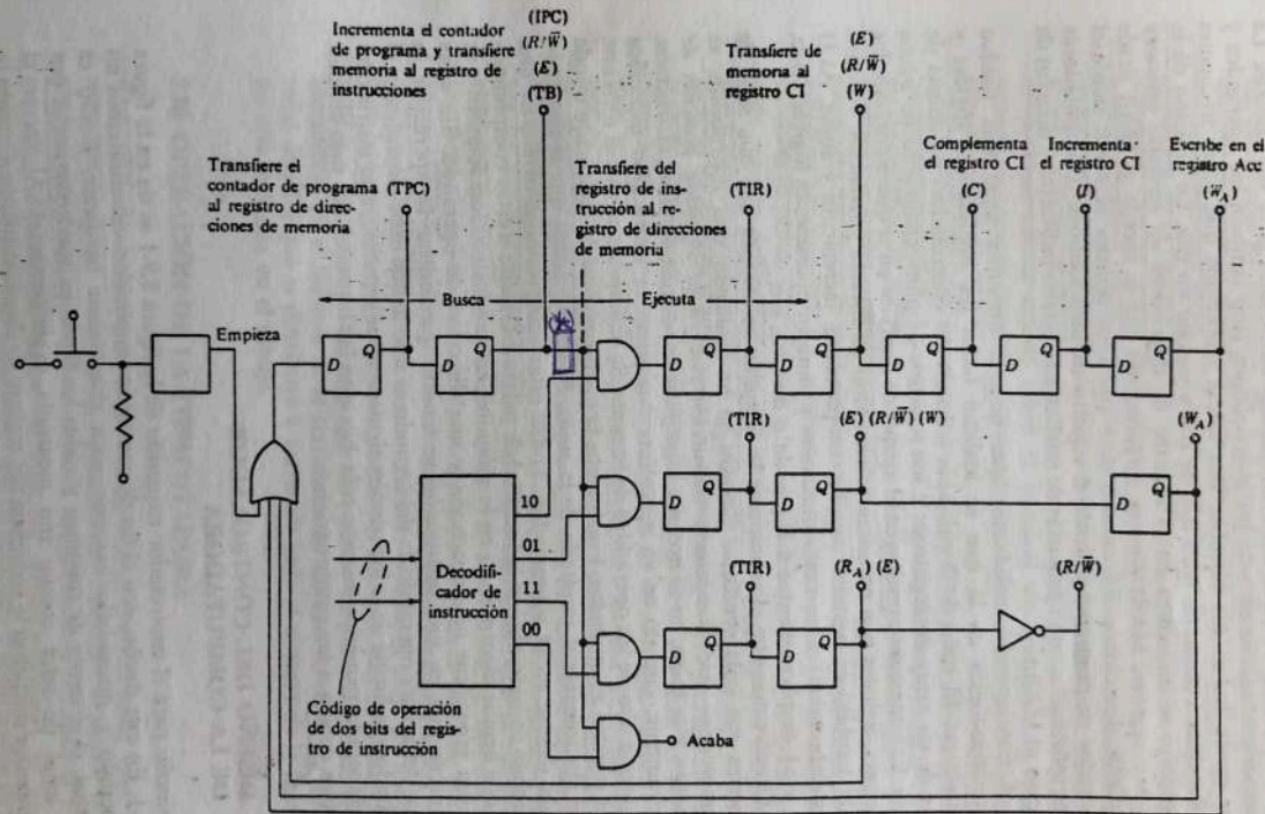


Figura 8.11-1 Controlador del «computador» de la figura 8.9-1.

salida de cada flip-flop consecutivo alcanza el 1 lógico, se realiza una microoperación. Hemos expresado en palabras que las microoperaciones se realizan en cada paso y tienen indicado entre paréntesis los terminales de control que deben activarse para que se puedan realizar. Los dos primeros pasos de la secuencia buscan la instrucción y la carga en el registro de instrucción. Esta parte de búsqueda de la operación del controlador es la misma sin importar la instrucción buscada. Una vez completada la parte de búsqueda de la instrucción comienza la parte de ejecución. La operación a controlar se transmite al controlador como un código de operación de 2 bits. En el controlador estos 2 bits se aplican a un decodificador con cuatro líneas de salida. Cuando el código de operación es 10 se indica la sustracción. Solamente la línea de salida 10 del decodificador está en 1 lógico y sólo se habilita la puerta AND del extremo superior. La secuencia del registro de desplazamiento continúa a través de la fila superior de flip-flops, disponiéndose de esta manera para realizar la secuencia apropiada para la resta. Una operación de código 01 habilita la segunda puerta AND de la parte superior. La secuencia resultante es la misma que para la sustracción excepto que se eliminan las microoperaciones de complementación e incrementación. El código 11 se utiliza para transferir el contenido del registro acumulador a memoria. Aquí se requieren dos microoperaciones. La primera, como en las operaciones de suma y resta, transfiere la dirección del registro de instrucción al registro de direcciones de memoria. La segunda escribe en memoria en vez de leer de memoria, por tanto, la inversión del nivel lógico se aplica al control R/W de memoria.

Si la instrucción es restar, sumar o transferir, la secuencia de ejecución cuando se completa nos lleva inmediatamente a través de la puerta OR al comienzo del ciclo de búsqueda. Entonces se busca la siguiente instrucción y comienza una nueva ejecución, etc. Si, por otra parte, la instrucción es 00, que significa parar, no hay retorno al ciclo de búsqueda. La operación cesa después que la puerta AND de la parte más inferior haya estado a 1 lógico durante un ciclo de reloj. Esta salida de «finalizado» puede utilizarse, si nos parece, para indicar que la computadora ha completado su trabajo y ahora se requiere intervención manual.

La parte de ejecución del controlador de la figura 8.11-1 utiliza más flip-flops que los estrictamente necesarios. Una parte de ejecución alternativa mostrada en la figura 8.11-2 es más económica en flip-flops aunque a expensas de utilizar más puertas. Se utilizan cinco flip-flops, permitiendo durante cada secuencia de ejecución cinco microoperaciones. En cada caso, excepto cuando la instrucción es alto, el primer paso de la secuencia se indica porque el terminal TIR alcanza el nivel activo. En el segundo paso, las operaciones de suma y resta tienen una microoperación común mientras la operación almacena, indica una microoperación alternativa. A menos que la instrucción sea de resta, los pasos tercero y cuarto de la secuencia se eliminan. Si se realiza la resta las microoperaciones tercera y cuarta son la de complemento e incremento. Finalmente, de nuevo, si se indica suma o resta hay una activación común de la entrada  $W_A$  para suministrar el quinto paso de la secuencia. En otros casos este quinto paso se elimina.

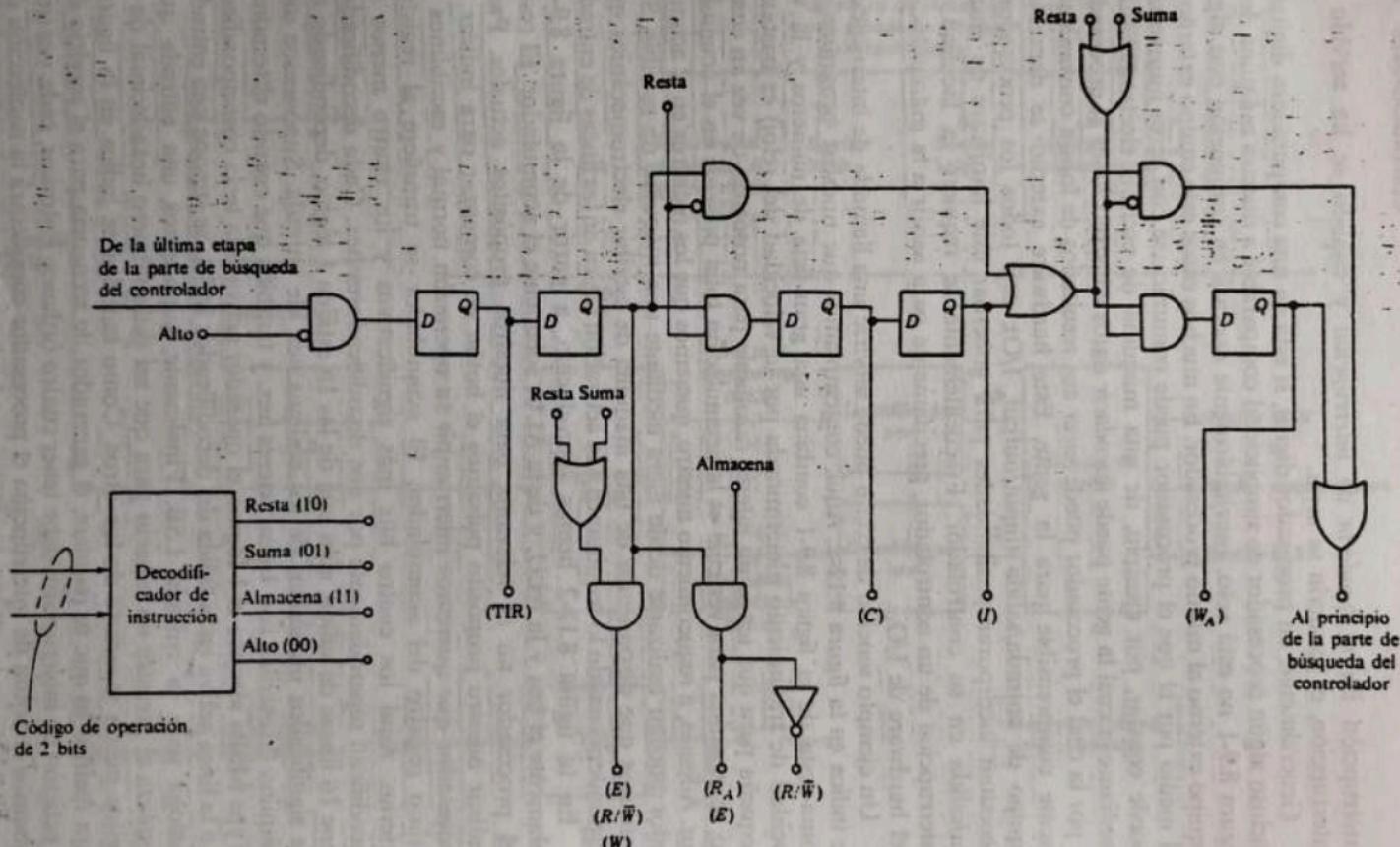


Figura 8.11-2 Controlador alternativo del controlador de la figura 8.11-1 que utiliza menos flip-flops pero más puertas.

## 8.12 INTERRUPCIONES

Una vez iniciado, el controlador de la figura 8.11-1 trabaja indefinidamente. Realiza una secuencia de búsqueda, después una de ejecución, vuelve a una secuencia de búsqueda, etc., hasta que finalmente encuentra una instrucción de alto que lo detiene. El controlador opera como el procesador digital de la figura 8.9-1, que no es capaz de hacer más que sumar y restar una columna de números. Aun así el modo de operación de este controlador (búsqueda, ejecución, búsqueda, etc.) es de fundamental importancia, ya que los controladores en los sistemas digitales más sofisticados operan del mismo modo.

Con frecuencia es necesario interrumpir este ciclo repetitivo búsqueda-ejecución e indicar al controlador que genere una secuencia de señales de órdenes de habilitación para realizar alguna operación especial. La fuente que requiere la operación especial se denomina llamada de interrupción y la respuesta correspondiente del controlador se denomina respuesta a una petición de servicio.

La figura 8.12-1 indica cómo puede aumentarse el controlador para incorporar la posibilidad de responder dicha petición de servicio. Si el cerrojo estático está en el estado de reset con  $Q = 1$  se habilita la puerta AND  $G_N$  para operación normal y está inhabilitada la puerta para operación de interrupción  $G_I$ . Una vez pulsado el botón de comienzo empezará una secuencia normal, primero a través de la parte de búsqueda del controlador, después a través de la ejecución, después vuelve a la búsqueda y así sucesivamente. Supongamos que en algún instante se recibe una señal indicando que se requiere una secuencia de servicio. La señal que llama la interrupción solamente necesita hacer una breve excursión al lógico aplicado a la entrada de set del cerrojo. Cuando éste está en set ( $Q = 1$ ,  $\bar{Q} = 0$ ), se habilita  $G_I$  en lugar de  $G_N$ . Supongamos que han comenzado los primeros pasos del ciclo de búsqueda antes que se reciba la señal de interrupción. Entonces, continuará la secuencia de búsqueda y será seguida por la de ejecución. Esto es, la instrucción que está en proceso de búsqueda y ejecución no será interrumpida pero cuando se haya completado la ejecución, como  $G_I$  está habilitada y  $G_N$  no, el secuenciamiento procederá a través del generador de secuencia auxiliar para el servicio de la interrupción. Cuando la secuencia procede así, se generan órdenes para activar las microoperaciones que se requieren. Hemos dispuesto que la última orden, además de realizar cualquier microoperación ponga también en reset al cerrojo. Así el controlador volverá a la operación normal.

El cerrojo en la figura 8.12-1 se necesita para almacenar la información de que alguna fuente ha hecho una petición de interrupción, ya que requiere un servicio. Se necesita este almacenamiento porque la señal de entrada de interrupción puede ser de corta duración y el controlador quizás no pueda responder hasta que haya completado la instrucción en proceso de realizarse. «Registros» de un bit, como el cerrojo, que no registran una palabra pero sirven para almacenar 1 bit de información para alertar al controlador de alguna situación, se denominan banderines (flags). Así la petición de una

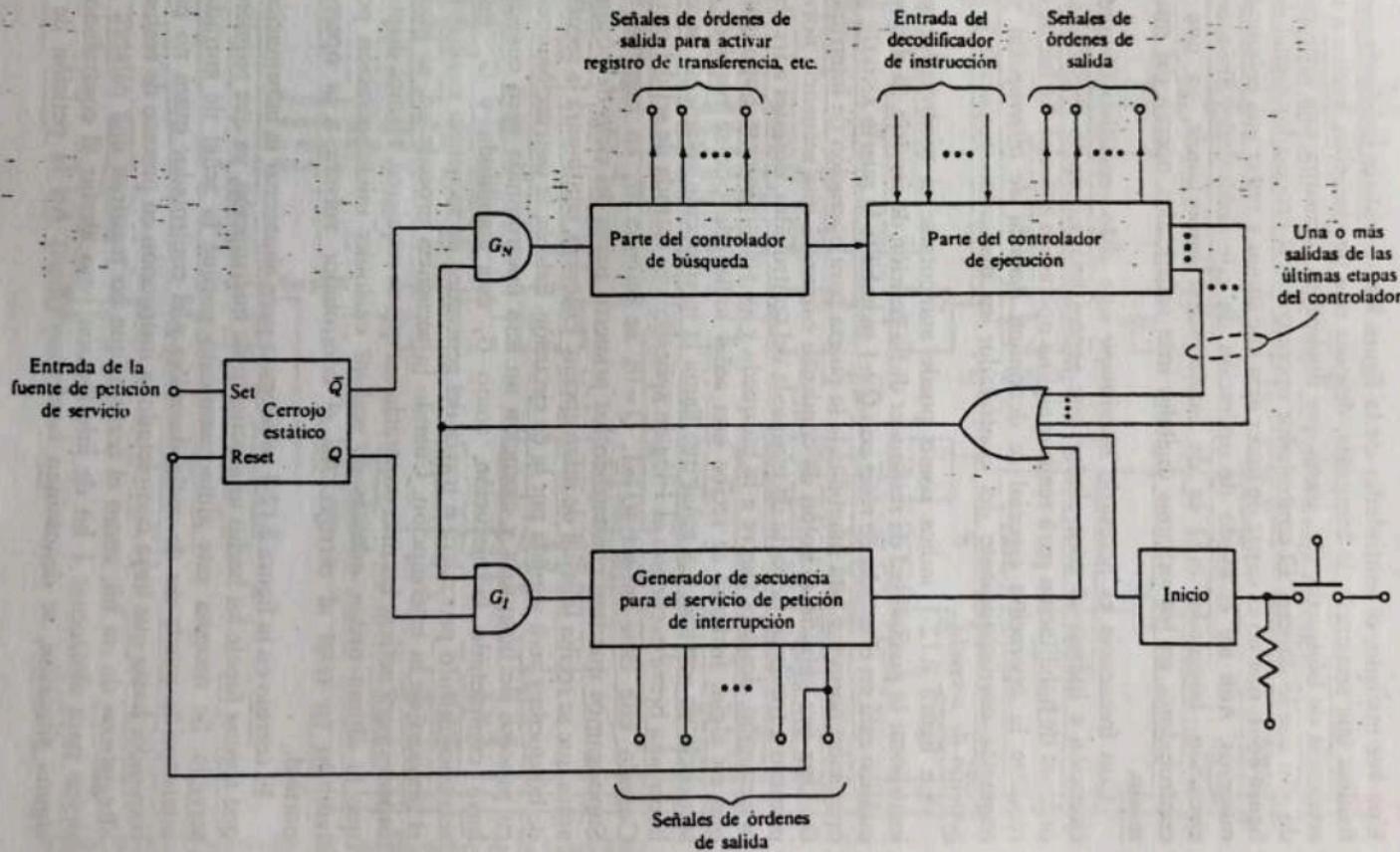


Figura 8.12-1 Modificación de controlador que permite responder a una petición de interrupción.

interrupcióniza el *banderín de interrupción* y cuando se ha servido la interrupción, el banderín se arría.

Generalmente, un procesador digital, si está en una computadora digital o incluso algún procesador de simplicidad comparable a nuestra máquina de la figura 8.9-1 no está solo. Inevitablemente operará en conjunción con algún equipo externo al mismo procesador. En muchos casos este equipo externo es el medio por el que el procesador puede comunicarse con seres humanos y puede constar, por ejemplo, de una máquina de escribir electromecánica mediante la cual la gente puede introducir datos e instrucciones al procesador y por la cual el procesador puede sacar sus resultados de forma convenientemente comprensible para la gente. Este hardware externo se denomina equipo de entrada/salida (input/output, I/O); casi todos los procesadores necesitan incorporar posibilidades para gobernar este hardware de entrada/salida en su controlador. Frecuentemente se requiere la facultad de interrupción de un controlador, precisamente para asistir a la manipulación del hardware de I/O.

Un ejemplo sencillo de cómo puede servirse una llamada de interrupción se indica en la figura 8.12-2. Aquí, contemplamos que nuestro programa de suma-resta de la figura 8.9-1 combina una gran lista de números y la velocidad de tratamiento (determinada por la velocidad del reloj) es bastante pequeña para que tenga un interés razonable para nosotros, de vez en cuando, se examina para ver cuál es la acumulación total presente en el acumulador. Además, a requerimiento nuestro, queremos que los registros en el acumulador puedan realizarse desde fuera mediante una máquina de escribir. Suponemos que disponemos de una máquina de escribir electromecánica que responderá cuando lo requiramos a la activación lógica de las líneas de entrada.

En la figura 8.12-2 hemos reproducido, a partir de la figura 8.9-1, solamente el bus y la ALU, y de la ALU, solamente el acumulador. El resto del procesador no es relevante para nuestros propósitos actuales. Para realizar nuestro propósito pulsamos el botón de comenzar para indicar al procesador que queremos interrumpir su operación normal y escribimos el último registro del acumulador. El acumulador se transfiere al registro externo. Aquí los cuatro bits más significativos y los cuatro menos se decodifican separadamente por dos decodificadores hex. Cada decodificador tiene 16 líneas de salida, sólo una de las 16 está en 1 lógico, dependiendo de los significados numéricos de los códigos hex. de entrada. Suponemos una máquina de escribir con 16 caracteres hex. y también que cuando el terminal «Type MSB» alcance el 1 lógico, el operador activará la tecla correspondiente a la línea activa en la salida del decodificador MSB. Se supone una entrada análoga para la entrada LSB. Finalmente, suponemos una entrada que desplaza el arrastre un espacio para que así cada número hexadecimal de 2 dígitos pueda separarse de los otros. Como también se indica en la figura, para realizar lo que se requiere, el generador de secuencia para el servicio de la petición de interrupción debe tener cuatro órdenes. Entonces, cada vez que pulsamos el botón de interrupción el procesador completará la confección de su instrucción presente e irá a la secuencia de servicio para atender nuestra petición.

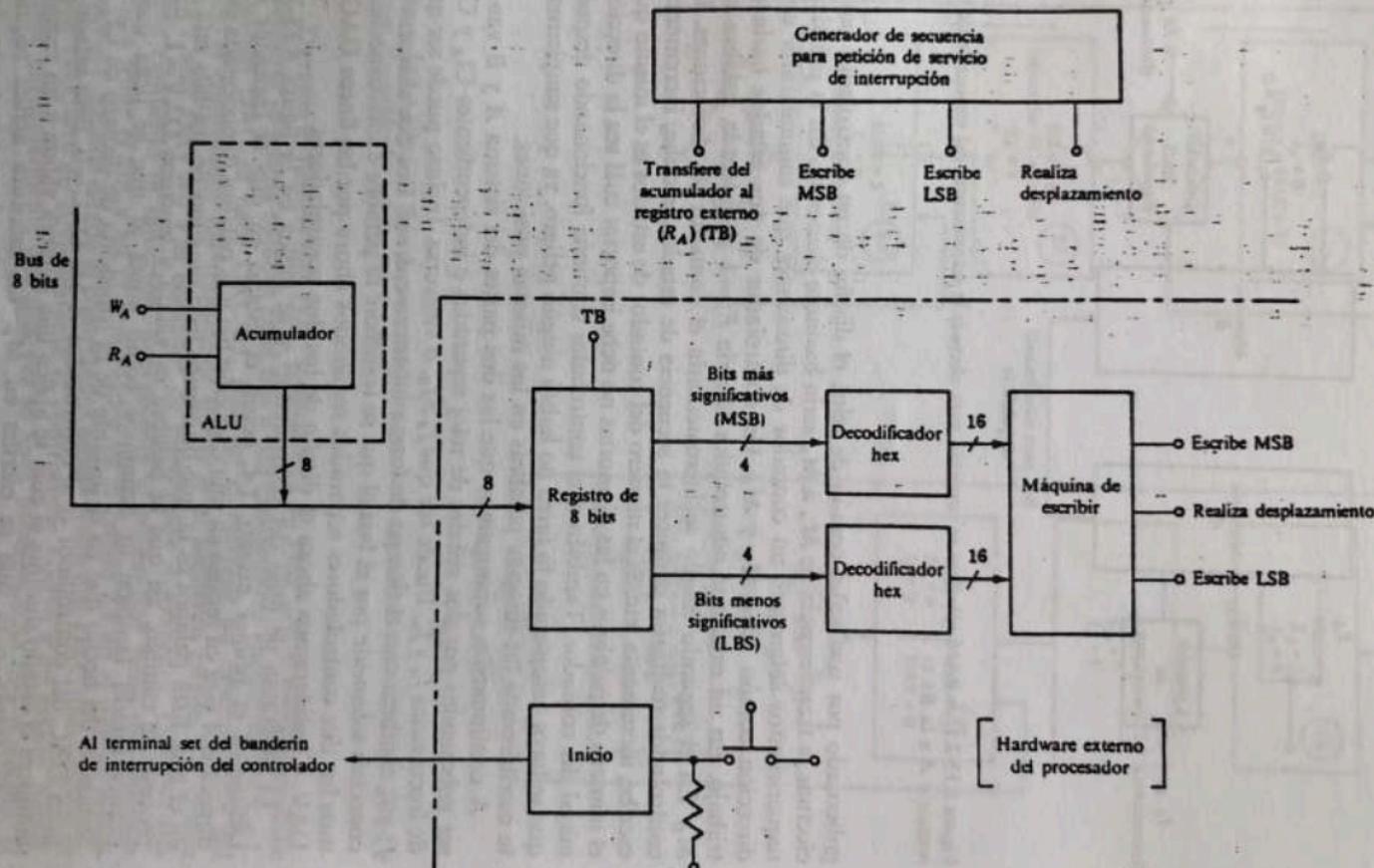


Figura 8.12-2 Ejemplo. Servicio de una interrupción que indica una lectura de salida del acumulador.

Aparece una dificultad muy clara en el esquema de la figura 8.11-2. Si construimos nuestro generador de secuencia como los ejemplos presentados antes, habremos permitido solamente un intervalo de reloj para cada operación (transferencia, escritura y desplazamiento). La transferencia del acumulador al registro externo se realiza fácilmente en un intervalo de reloj, pero a menos que la velocidad de reloj sea muy lenta, un intervalo de reloj no será adecuado para las respuestas mecánicas requeridas de la máquina de escribir. Una forma de resolver dificultades debidas a la lenta respuesta es aumentar de uno a muchos el número de flip-flops del generador de secuencia entre dos puntos de los que se sirven las órdenes. Este método es, por supuesto, caro en hardware. En cualquier circunstancia se recurre regularmente a este problema de hacer interfaces entre sistemas que no operan a la misma velocidad y en la siguiente sección, nos referimos a este problema.

### 8.13 PRESENTACIÓN (HANDSHAKING)

Un método por el que podemos lograr comunicación entre dos sistemas que operan asíncronamente y a menudo a velocidades inconexas y bastante diferentes se denomina presentación (handshaking). Consideraremos el problema específicamente en conexión con la transmisión de datos de un sistema transmisor a uno receptor.

En el reconocimiento, el transmisor tiene disponible para el receptor, además de los datos a transmitir, una línea que conlleva una señal lógica denominada datos válidos (data valid, DAV). El receptor tiene disponible para el transmisor otra línea que conlleva una señal lógica denominada datos aceptados (data accepted, DAC). La disposición se ilustra en la figura 8.13-1. Presumiblemente el transmisor transmitirá al receptor palabras sobre un bus de datos de  $n$ -bits. Cuando  $DAV=1$ , significa que cualquier tratamiento y operaciones que necesiten realizarse para colocar una nueva palabra en el bus de datos se ha completado y la palabra en el bus está establecida y estable y preparada para ser leída por el receptor; es decir, la palabra es válida. Cuando  $DAV=0$ , la palabra no es válida. Una secuencia de niveles lógicos sobre DAV de 1 a 0 y vuelta a 1 acompañan cada cambio de palabra.

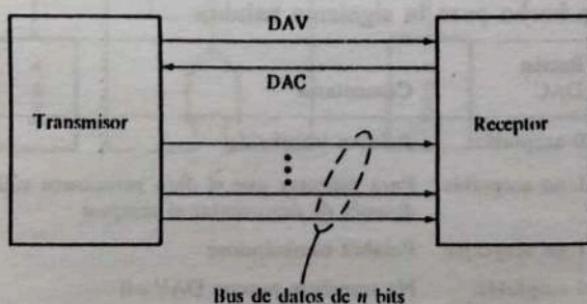


Figura 8.13-1 La operación de un transmisor y receptor de datos se coordina por el uso de las líneas de presentación (handshaking) DAV y DAC.

Cuando  $DAC = 0$ , significa que el receptor está completamente preparado para recibir la palabra de datos del bus pero hará esto solamente si encuentra que  $DAV = 1$ . Si  $DAC = 1$ , el receptor no aceptará el dato, es decir, el receptor está, en efecto, desconectado del bus. Transmisor y receptor se comunican sobre estas líneas  $DAV$  y  $DAC$ . El receptor llama al transmisor cuando se ha aceptado la palabra de datos para que éste pueda poner la siguiente palabra en la linea. El transmisor llama al receptor cuando una nueva palabra está estable en el bus para que pueda ser leída.

Supongamos que comenzamos durante un intervalo cuando  $DAV = 1$  y  $DAC = 0$ , esto es, se está transmitiendo una palabra y se está en el proceso de que sea aceptada. Entonces debemos suponer que  $DAV$  no llega a hacerse  $DAV = 0$  hasta que tengamos  $DAC = 1$ . También cuando con  $DAV = 1$  y  $DAC = 0$  se acepta una palabra, no debe aceptarse una nueva palabra hasta que  $DAV$  cambie a  $DAV = 0$  y posteriormente vuelva de nuevo a  $DAV = 1$ . En cualquier otro caso, el receptor aceptará la misma palabra dos veces. Con dicho sistema seguramente habrá intervalos en los que el transmisor deba esperar al receptor para completar algún tratamiento o cuando el receptor deba esperar al transmisor. En la práctica, lo importante es asegurar que el transmisor o receptor esperarán lo que se requiera. La secuencia que siguen las líneas  $DAV$  y  $DAC$  cuando se transfiere una palabra y se hacen preparativos para transmitir la siguiente, se muestra en la tabla 8.13-1.

Como ilustración del uso de las líneas  $DAV$  y  $DAC$ , consideremos la situación de la figura 8.13-2 donde tenemos dos memorias de acceso aleatorio de organización idéntica y nuestro propósito es transferir el contenido de la memoria  $A$  ( $M_A$ ) a la memoria  $B$  ( $M_B$ ). La palabra ocupará en  $B$  la misma dirección que tenía en  $A$ . Permanentemente hemos puesto la entrada de habilitación  $E_A$  y  $(R/W)_A$  en  $E_A = (R/W)_A = 1$  para que haya una salida de palabra de  $M_A$ . La palabra se determina por un contador cuya cuenta indica la dirección de la palabra seleccionada. En  $M_B$  hemos puesto  $(R/W)_B = 0$  para que  $M_B$  esté permanentemente en el modo de escritura y esta operación ocurrirá cuando el terminal de habilitación vaya a  $E_B = 1$ .

Si el sistema completo operase como un sistema sincrónico, es decir,

**Tabla 8.13-1 Secuencia de los niveles lógicos  $DAV$  y  $DAC$  cuando se transmite una palabra y preparación hecha para la siguiente palabra**

Intervalo de tiempo	Transfiere $DAV$	Recibe $DAC$	Comentario
1	1 válido	0 aceptable	Palabra transferida
2	1 válido	1 no aceptable	Para asegurar que el dato permanece válido después de desconectar el receptor
3	0 no válido	1 no aceptable	Palabra cambiándose
4	0 no válido	0 aceptable	No transfiere porque $DAV = 0$
5	1	0	Nueva palabra transferida

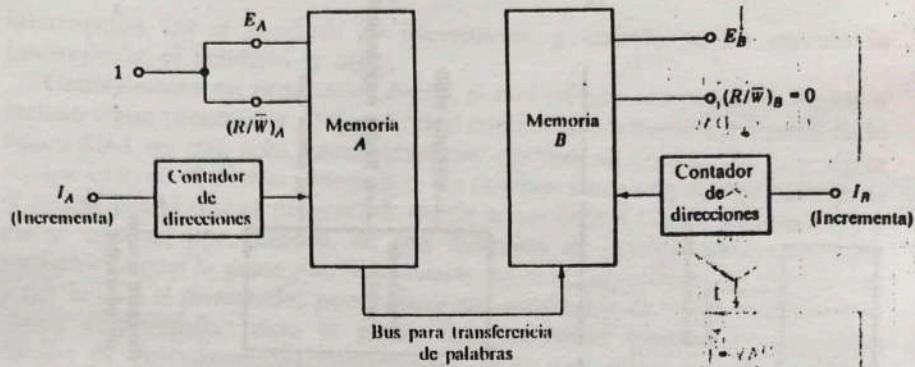


Figura 8.13-2 Dos RAM idénticas se conectan para efectuar la transferencia del contenido de la memoria  $A$  a la  $B$ .

gobernado por una señal común de reloj, el diseño de un controlador para efectuar la transferencia de  $M_A$  a  $M_B$  sería bastante elemental. En la práctica, requeriríamos solamente un contador de direcciones que suministraría una dirección común para  $M_A$  y  $M_B$ . Un controlador de dos estados haría el trabajo. En un estado el controlador tendría  $E_B=1$  para leer la palabra en  $M_B$ . En el segundo estado se incrementaría el contador de direcciones. El controlador realizaría entonces la secuencia de estados, escribe, incrementa, escribe, incrementa, etc. Si el número del contador de estados es el mismo que el número de palabras en las memorias no debe importar cuál sea la dirección inicial del contador. También si el controlador continúa funcionando después que se haya completado la tarea no habrá ningún peligro, ya que simplemente escribiremos las mismas palabras en las mismas posiciones.

A continuación supongamos que las dos partes del sistema  $A$  y  $B$  van a ser gobernadas por dos señales de reloj separadas e independientes  $Cl_A$  y  $Cl_B$  de frecuencias  $f_A$  y  $f_B$ . Puede ser que  $f_A > f_B$ , o viceversa. Incluso puede ser que  $f_A$  y  $f_B$  cambien con el tiempo de forma inconexa entre sí. Los dos sistemas se conectan *sólo* por el bus al que se transfiere la palabra e interconectaremos los dos controladores separados requeridos ahora por las líneas DAC y DAV. Consideramos ahora el diseño de los dos controladores.

El diagrama de flujo del controlador  $A$  se indica en la figura 8.13-3a. Definimos el primer estado,  $I_A$ , como el estado en que el contador de dirección  $A$  está en reposo en una dirección fija y la palabra direccionada está en el bus. Esta palabra es la que queremos leer; ninguno de sus bits está en proceso de cambio, así que la palabra es válida y tenemos  $DAV=1$ . Este primer estado también lo definimos como aquel al que se llega como resultado del hecho que en el instante de la transición de disparo de  $Cl_A$ , la línea DAC del controlador  $B$  al controlador  $A$  era  $DAC=0$ . Este estado es entonces el estado durante el cual se aplica una palabra válida a un receptor que la acepta para que se escriba en  $M_B$ . La escritura actual requiere solamente que el controlador coloque  $E_B$  en 1 lógico durante este estado.

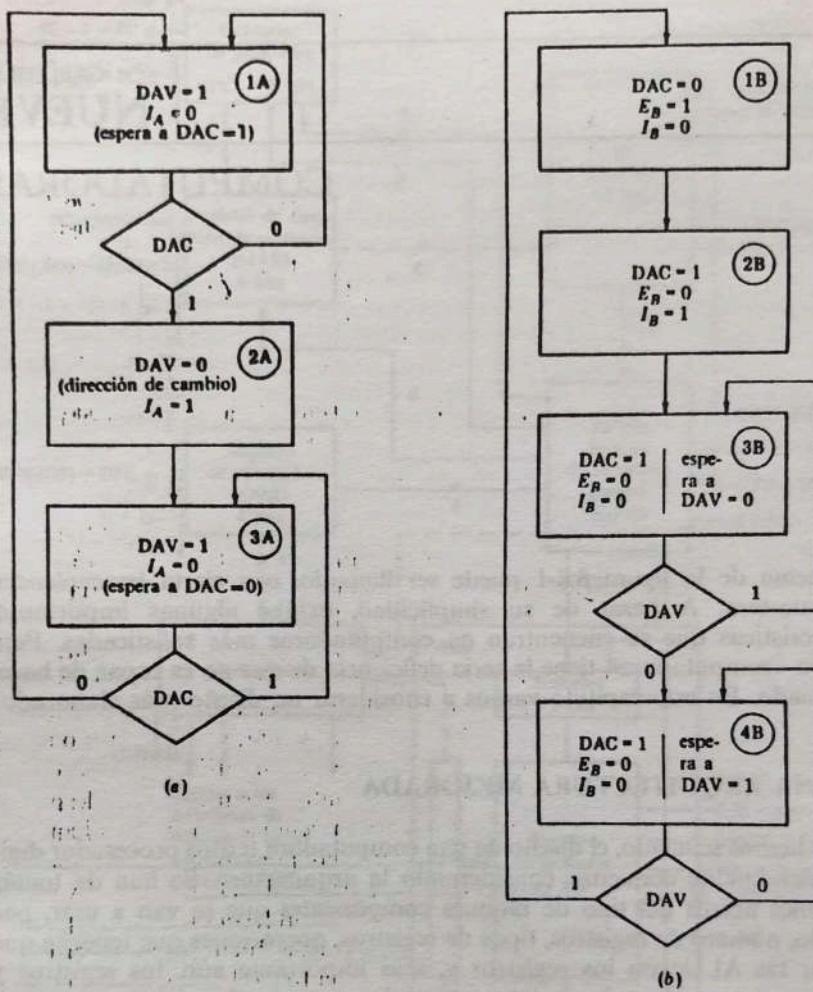


Figura 8.13-3 Carta de flujo de un controlador para que opere el sistema de la figura 8.13-2 y tenga disponibles las señales DAV y DAC.

Durante el tiempo que DAC permanezca en  $DAC=0$  el transmisor debe encontrar la misma palabra en el bus y por consiguiente encuentra  $DAV=1$ . Cuando DAC se hace  $DAC=1$ , esto es, el receptor ha desconectado del bus, el controlador del transmisor puede ir al siguiente estado 2A, donde DAV será  $DAV=0$  y el contador de direcciones A avanzará. Después que haya cambiado la dirección, nuevamente el transmisor tiene una palabra válida sobre el bus y así el controlador va al estado 3A en el que de nuevo  $DAV=1$ . En este estado el controlador espera hasta que DAC nuevamente se haga  $DAC=0$ , en cuyo instante el controlador vuelve a su estado inicial. Notar particularmente que una vez se haya incrementado una dirección, el

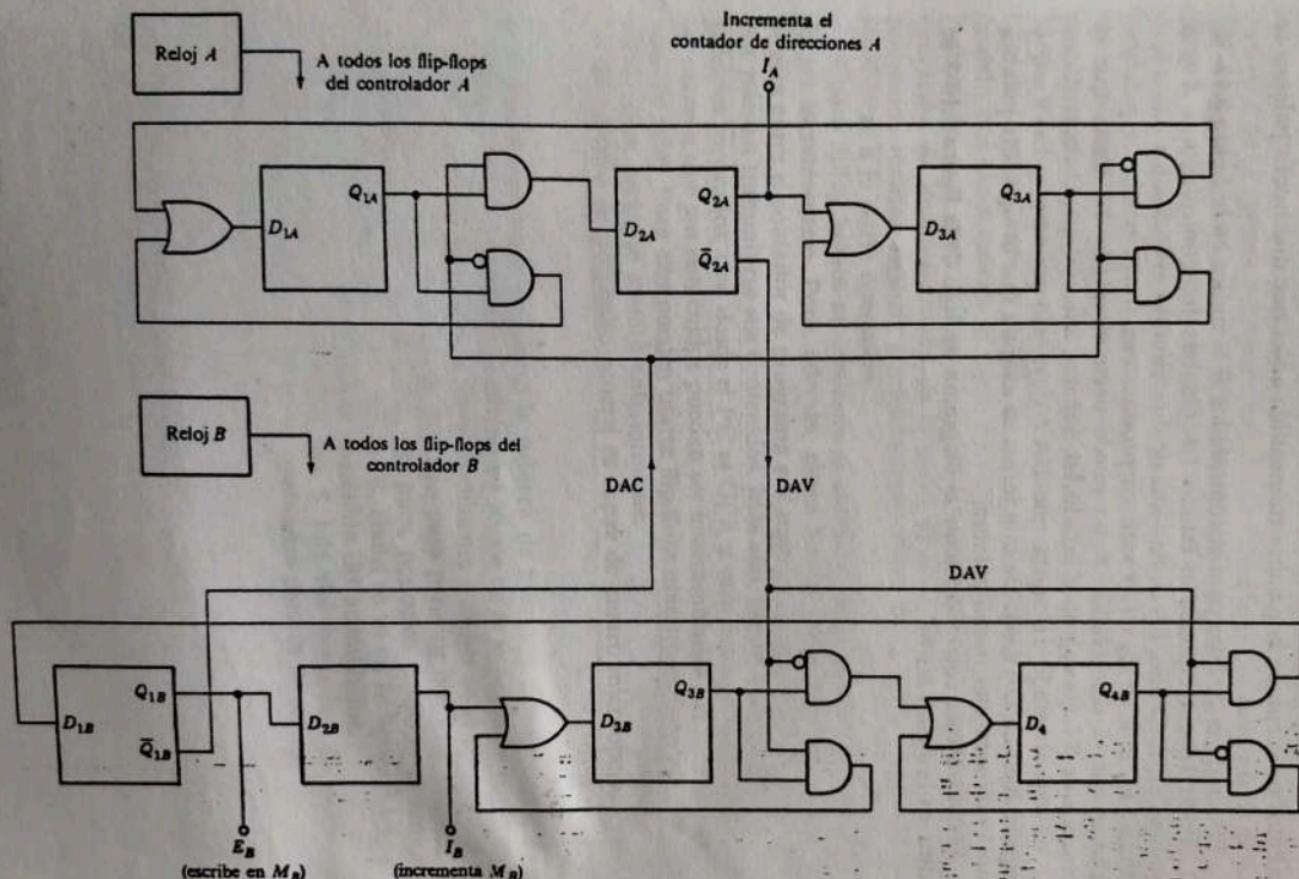


Figura 8.13-4 Controlador de registro de desplazamiento correspondiente al diagrama de flujo de la figura 8.13-3. Se indican las líneas de interconexión DAV y DAC.

siguiente cambio de dirección no ocurrirá hasta que DAC se haga primero  $DAC=0$  y después  $DAC=1$ . De esta forma el controlador hace cierto que el receptor ha aceptado la palabra transmitida antes que una nueva palabra se coloque en el bus.

El diagrama de flujo para el controlador  $B$  se indica en la figura 8.13-3b. El estado  $1B$  corresponde al estado  $1A$ . Cuando los controladores  $A$  y  $B$  están en estos estados, los contadores de dirección no avanzan, los datos son válidos.  $E_B$  está en  $E_B=1$ , y está en proceso una transferencia de palabra. Señalar que el controlador  $A$  no puede dejar el estado  $1A$  hasta que el controlador  $B$  haya dejado el estado  $1B$ . Señalar, además, que el controlador no volverá al estado  $1B$  hasta que DAV haya ido primero a  $DAV=0$  y después vuelto a  $DAV=1$ . De esta forma se asegura que la siguiente palabra transferida será una nueva palabra.

El controlador correspondiente al diagrama de flujo de la figura 8.13-3 se indica en la figura 8.13-4.