Lógica de trasferencia entre registros

8

8-1 INTRODUCCION

Un sistema digital es un sistema lógico secuencial construido con flip-flops y compuertas. Se ha mostrado en los capítulos anteriores que un circuito secuencial puede ser especificado por medio de la tabla de estado. Para especificar un sistema digital extenso, con una tabla de estado, sería muy difícil, si no imposible, porque el número de estados sería demasiado grande. Para sobreponer esta dificultad, se diseñan invariablemente los sistemas digitales usando una alternativa modular. El sistema se subdivide en subsistemas modulares, cada uno de los cuales realiza algún trabajo funcional. Los módulos se construyen a partir de funciones digitales tales como registros, contadores, decodificadores, multiplexores, elementos aritméticos y lógica de control. Los diferentes módulos se interconectan con datos comunes de control para formar un sistema de computador digital. Un módulo sistema digital típico sería la unidad procesadora de un computador digital.

La interconexión de las funciones digitales para formar un módulo sistema digital no puede describirse por medio de técnicas combinacionales o de secuencias lógicas. Estas técnicas fueron desarrolladas para describir un sistema digital a nivel de compuerta y flip-flop y no son apropiadas para describir el sistema a nivel de función digital. Para describir un sistema digital en términos de funciones tales como sumadores, decodificadores y registros, es necesario emplear una notación matemática de alto nivel. El método de lógica de trasferencia entre registros copa esta necesidad. En este método, se seleccionan registros como los componentes primitivos de un sistema digital en vez de las compuertas y los flip-flops como en la lógica secuencial. En esta forma es posible describir de una manera precisa y concisa el flujo de información y las tareas de procesamiento entre los datos acumulados en los registros. La lógica de trasferencia de registros usa un conjunto de expresiones y afirmaciones, las cuales tienen una similitud con las afirmaciones usadas en los lenguajes de programación. Esta notación presenta las herramientas necesarias para especificar un conjunto prescrito de interconexiones entre varias funciones digitales.

Una característica importante de presentación del método lógico de trasferencia entre registros es que está relacionado muy de cerca a la forma en que la genta prefiere especificar las operaciones del sistema digital.

Los componentes básicos de este método son aquellos que describen un sistema digital a partir del nivel operacional. La operación de un sistema digital se describe de mejor manera especificando:

1. El conjunto de registros en el sistema y sus funciones.
2. La información en código binario almacenada en los registros.
3. Las operaciones realizadas a partir de la información almacenada en los registros.
4. Las funciones de control que inician la secuencia de operaciones.

Estos cuatro componentes forman la base del método de lógica de trasferencia entre registros para describir sistemas digitales.

Un *registro* como se define en la notación de lógica de trasferencia entre registros, no solamente implica un registro, parecido al definido en el Capítulo 7, si no que abarca también todos los otros tipos de registros, tales como registros de desplazamiento, contadores y unidades de memoria. Un contador se considera como un registro cuya función es incrementar en 1 la información almacenada en él. Una unidad de memoria se considera como una colección de registros de almacenamiento donde se va a almacenar la información. Un flip-flop por si solo se toma como un registro de 1 bit. De hecho, los flip-flops y las compuertas asociadas de cualquier circuito secuencial se llaman registro, al usar este método de designación.

La *información binaria* almacenada en los registros podría ser números binarios, números decimales binarios codificados, caracteres alfanuméricos, control de información ó cualquier información binaria codificada. Las operaciones que se realizan mediante los datos almacenados en los registros, depende del tipo de datos que se encuentren. Los números se manipulan con operaciones aritméticas, mientras que el control de información se manipula por lo general con operaciones lógicas tales como activando o borrando bits específicos del registro.

Las operaciones realizadas con los datos almacenados en los registros se llaman *microoperaciones*. Una microoperación es una operación elemental que puede ser realizada en paralelo durante un período de pulso de reloj. El resultado de la operación puede remplazar la información binaria previa de un registro o puede ser trasferido a otro registro. Ejemplos de microoperaciones son: desplazar, contar, sumar, borrar y cargar. Las funciones digitales introducidas en el Capítulo 7 son registros que configuran microoperaciones. Un contador con carga en paralelo es capaz de realizar el incremento de las microoperaciones y la carga. Un registro de desplazamiento bidireccional es apto para realizar microoperaciones de desplazamiento a la derecha o a la izquierda. Las funciones MSI combinacionales, introducidas en el Capítulo 5 pueden ser usadas en algunas aplicaciones para realizar microoperaciones. Un sumador binario en paralelo es útil para realizar la microoperación de *suma* (add) a partir de los contenidos de los dos registros que retienen números binarios. Una microoperación requiere

solamente un pulso de reloj para su ejecución, si se hace la operación en paralelo. En los computadores en serie, una microoperación requiere un número de pulsos igual al tiempo de palabra en el sistema. Este último es igual al número de bits en los registros de desplazamiento que trasfieren la información en serie mientras que la microoperación se ejecuta.

Las *funciones de control* que inician la secuencia de operaciones consisten de señales de tiempo que le dan secuencia a las operaciones una por una. Ciertas condiciones que dependen de los resultados de las operaciones previas pueden determinar también el estado de las funciones de control. Una función de control es una variable binaria que en un estado binario inicia una operación y en el otro inhibe la operación.

El propósito de este capítulo es introducir en detalle los componentes del método de lógica de trasferencia entre registros. El capítulo introduce una notación simbólica para representar registros, para operaciones específicas en los contenidos de los registros y para especificar funciones de control. Esta notación simbólica se llama algunas veces *lenguaje de trasferencia entre registros* o *lenguaje descriptivo de material del computador* (register-transfer language or computer hardware description language). El lenguaje de trasferencia entre registros adoptado aquí pretende ser el más sencillo posible. Debe tenerse en cuenta, sin embargo, que no existe simbología normalizada para el lenguaje de trasferencia entre registros y fuentes diferentes adoptan convenciones diferentes.

Una afirmación en un lenguaje de trasferencia entre registros consiste de una función de control y una lista de microoperaciones. La función de control (la cual puede ser omitida algunas veces) especifica la condición de control y secuencia de tiempos para ejecutar la lista de microoperaciones. Las microoperaciones especifican las operaciones elementales que se realizan con la información almacenada en los registros. Los tipos de microoperaciones encontradas más a menudo en los sistemas digitales pueden clasificarse en cuatro categorías:

1. Microoperaciones de *trasferencia entre registros* que no cambian el contenido de la información cuando la información binaria se mueve de un registro a otro.
2. Las microoperaciones *aritméticas* realizan aritmética con los números almacenados en los registros.
3. Las microoperaciones *lógicas* realizan operaciones tales como AND y OR con el par de bits individuales almacenados en los registros.
4. Las microoperaciones de *desplazamiento* especifican operaciones para los registros de desplazamiento.

Las Secciones 8-2 hasta 8-4 definen un conjunto básico de microoperaciones. Se asignan símbolos especiales a las microoperaciones en el conjunto y cada símbolo se muestra asociado con los materiales digitales correspondientes que configuran la microoperación establecida. Es importante tener en cuenta que la notación lógica de trasferencia entre registros se relaciona directamente con los registros y las funciones digitales que esta define y no pueden separarse de ellos.

Las microoperaciones ralizadas con la operación almacenada en los registros depende del tipo de datos que residen en los registros. La información binaria encontrada comúnmente en los registros de los computadores digitales puede clasificarse en tres categorías:

1. Datos numéricos tales como números binario o decimales binarios codificados usados en los cálculos aritméticos.
2. Datos no numéricos tales como caracteres alfanuméricos u otros símbolos binarios codificados usados en aplicaciones especiales.
3. Códigos de instrucciones, direcciones y otra información de control usada para especificar los requerimientos de procesamiento de datos del sistema.

Las Secciones 8-5 hasta 8-9 tratan sobre la representación de datos numéricos y su relación con las microoperaciones aritméticas. La Sección 8-10 explica el uso de las microoperaciones lógicas para el procesamiento de datos no numéricos. La representación de los códigos de instrucción y su manipulación con microoperaciones, se presenta en las Secciones 8-11 y 8-12.

8-2 TRASFERENCIA ENTRE REGISTROS

Los registros de un sistema digital son designados por letras mayúsculas (algunas veces seguidas de números) para denotar la función del registro. Por ejemplo, el registro que retiene una dirección para la unidad de memoria se llama comúnmente registro de direcciones de memoria y se designa como *MAR* (memory address register). Otras designaciones para el registro son *A*, *B*, *R*1, *R*2 e *IR*. Las celdas o flip-flops de un registro de n bits se numeran en secuencia desde 1 hasta n (o desde 0 hasta $n-1$) comenzando desde la izquierda o desde la derecha. La Figura 8-1 muestra cuatro maneras de representar un registro en la forma de diagrama de bloque. La forma más común de representar un registro es por medio de un rectángulo con el nombre del registro dentro de él de la manera mostrada en la Figura 8-1(a). Las celdas individuales pueden ser distinguidas como en (b), cada celda con su respectiva letra y número suscrito. La numeración de las celdas de derecha a izquierda puede ser marcada en la parte superior del rectángulo como en el registro *MBR* de 12 bits en (c). Un registro de 16

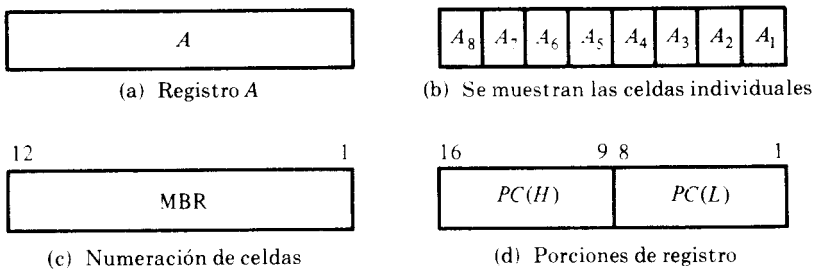


Figura 8-1 Diagrama de bloque de los registros

bits se divide en dos partes en (d). Los bits 1 a 8 se designan por medio de la letra L (viene de low) y los bits 9 a 16 se les asigna la letra H (viene de high). El nombre del registro de 16 bits es PC. El símbolo $PC(H)$ se refiere a las ocho celdas de mayor orden y $PC(L)$ se refiere a las ocho celdas de menor orden del registro.

Los registros pueden especificarse en el lenguaje de trasferencia entre registros con una afirmación declaratoria. Por ejemplo, los registros de la Figura 8-1 pueden definirse con las afirmaciones declaratorias tales como:

DECLARE REGISTER $A(8)$, $MBR(12)$, $PC(16)$
DECLARE SUBREGISTER $PC(L) = PC(1\text{–}8)$, $PC(H) = PC(9\text{–}16)$

Sin embargo, en este libro no se usarán proposiciones de declaración para definir los registros; en vez de ello los registros se mostrarán en la forma de diagrama de bloque como en la Figura 8-1. Los registros mostrados en un diagrama de bloque pueden convertirse fácilmente en proposiciones de declaración para propósitos de simulación.

La trasferencia de información de un registro a otro se designa en forma simbólica por medio del *operador de remplazo*. La proposición:

$$A \leftarrow B$$

denota la trasferencia del *contenido* del registro B al registro A. Esta designa un remplazo del contenido de A por lo contenido en B. Por definición, lo contenido en el registro fuente B no cambia después de la trasferencia.

Una proposición que especifica una trasferencia entre registros implica que los circuitos están conectados entre las salidas del registro fuente hasta las celdas de entrada del registro de destino. Normalmente no se requiere que ocurra esta trasferencia con cada pulso de reloj, sino solamente bajo una condición predeterminada. La condición que determina cuando ocurre la trasferencia se llama *función de control*. Una función de control es una función de Boole que puede ser igual a 1 ó 0. La función de control se incluye en la proposición como sigue:

$$x'T_1\colon \quad A \leftarrow B$$

La función de control se determina con dos puntos. Esta simboliza las necesidades que la operación de trasferencia puede ejecutar por medio de los materiales, solamente cuando la función de Boole $x'T_1 = 1$, es decir, cuando la variable $x = 0$ y la variable de tiempo $T_1 = 1$.

Cada proposición escrita en el lenguaje de trasferencia de registros implica una construcción con materiales para configurar la trasferencia. La Figura 8-2 muestra la configuración para la proposición escrita anteriormente. Las salidas del registro B se conectan a las entradas del registro A, y el número de líneas en esta condición es igual al número de bits en los registros. El registro A debe tener una entrada de control de carga de tal manera que pueda habilitarse cuando la función de control es 1.

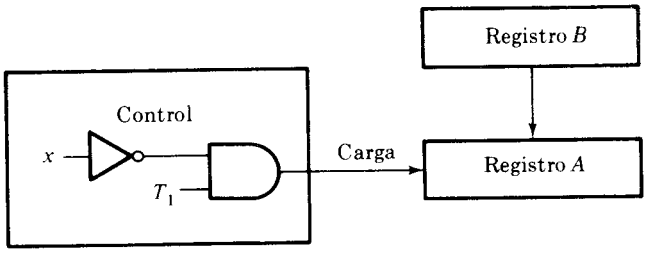


Figura 8-2 Configuración con componentes de la proposición $x'T_1$: $A \leftarrow B$

Aunque no se muestra, se asume que el registro A tiene una entrada adicional que acepta pulsos continuos sincronizados de reloj. La función de control se genera por medio de un inversor y una compuerta AND. Se asume también que la unidad de control que genera la variable de tiempo T_1 se sincroniza con los mismos pulsos de reloj que se aplican al registro A. La función de control permanece activa durante un período de pulso de reloj (cuando la variable de tiempo es igual a 1) y la trasferencia ocurre durante la siguiente transición de un pulso de reloj.

Los símbolos básicos de la lógica de trasferencia de registros se listan en la Tabla 8-1. Los registros se denotan por letras mayúsculas y los números pueden estar contiguos a las letras. Los suscritos se usan para distinguir las celdas individuales del registro. Los paréntesis se usan para definir una porción de un registro. La flecha denota una trasferencia de información y la dirección de la misma. Dos puntos terminan una función de control y la coma se usa para separar dos o más operaciones que se ejecutan al mismo tiempo. La proposición:

$$T_3\colon \quad A \leftarrow B, \qquad B \leftarrow A$$

denota una operación de intercambio que trasfiere los contenidos de dos registros durante un pulso de reloj común. Esta operación simultánea es posible en los registros con flip-flops maestro esclavo o por disparo de flanco.

Las llaves cuadradas se usan conjuntamente con la trasferencia de memoria. La letra M designa una palabra de memoria y el registro encerrado dentro de las llaves cuadradas significa la dirección para la memoria. Esto se explica en más detalle a continuación.

Hay ocasiones cuando el registro de destino recibe información de dos fuentes pero evidentemente no al mismo tiempo. Considérese dos proposiciones:

$$T_1\colon \quad C \leftarrow A$$

$$T_5\colon \quad C \leftarrow B$$

La primera línea establece que el contenido del registro A va a ser trasferido al registro C cuando ocurre una variable de tiempo T_1. La segunda proposición usa el mismo registro de destino que la primera pero con un

Tabla 8-1 Símbolos básicos de la lógica de trasferencia entre registros

Símbolo	Descripción	Ejemplos
Letras (y numerales)	Denota un registro	*A, MBR, R2*
Suscrito	Denota un bit de un registro	A_2, B_6
Paréntesis ( )	Denota una porción de un registro	*PC(H), MBR(OP)*
Flecha ←	Denota una trasferencia de información	$A \leftarrow B$
Dos puntos :	Termina una función de control	$x'T_0$:
Coma ,	Separa dos microoperaciones	$A \leftarrow B, B \leftarrow A$
Llaves cuadradas [ ]	Especifica una dirección para una trasferencia de memoria	$MBR \leftarrow M[MAR]$

registro fuente diferente y una variable de tiempo diferente. La conexión de dos registros fuente al mismo registro de destino no puede hacerse directamente, pero requiere un circuito multiplexor para seleccionar entre dos caminos posibles. El diagrama de bloque del circuito que configura las dos proposiciones se muestra en la Figura 8-3. Para registros con cuatro bits cada uno, se necesita un multiplexor de 2 a 1 líneas cuádruple, similar al mostrado previamente en la Figura 5-17 para seleccionar el registro A o el registro B. Cuando $T_5 = 1$ se selecciona el registro B, pero cuando $T_1 = 1$ se selecciona el registro A (porque T_5 debe ser 0 cuando T_1 es 1). El multiplexor y la entrada de carga del registro C se habilita cada vez que ocurra T_1 ó T_5. Esto causa una trasferencia de información del registro fuente seleccionado al registro de destino.

Bus de trasferencia

A menudo un sistema digital tiene muchos registros y se debe proveer de caminos para trasferir información de un registro a otro. Considérese por

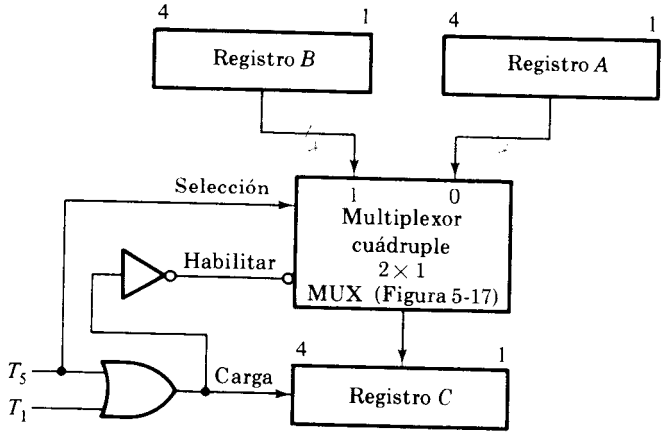


Figura 8-3 Uso de un multiplexor para trasferir información de dos fuentes a un solo destino

ejemplo los requerimientos de trasferencia entre los tres registros como se muestra en la Figura 8-4. Hay seis líneas de datos y cada registro requiere un multiplexor para seleccionar entre dos fuentes. Si cada registro consiste de n flip-flops, hay necesidad de 6 n líneas y tres multiplexores. A medida que aumenta el número de registros, aumenta el número de multiplexores y el número de líneas de interconexión. Si se restringe la trasferencia a uno a uno, el número de caminos entre los registros pueden reducirse considerablemente. Esto se muestra en la Figura 8-5, donde la salida y entrada de cada flip-flop se conecta a la línea común a través de un circuito electrónico que actúa como un interruptor. Todos los interruptores están abiertos normalmente hasta que se requiera una trasferencia. Para una trasferencia de F_1 a F_3, por ejemplo, se cierran los interruptores S_1 y S_4 para formar el camino requerido. El esquema puede ser extendido a los registros con n flip-flops, y este requiere n líneas comunes.

Un grupo de alambres a través de los cuales se trasfiere la información binaria bit a bit, un bit a la vez entre registros se llama *bus*. Para la trasferencia en paralelo, el número de líneas en el bus es igual al número de bits en los registros. La idea de un bus de trasferencia es análoga al sistema de trasporte central usado para llevar gente de un lado para el otro. En vez de que cada viajero use trasporte privado para ir de un lugar a otro, se usa un sistema de bus y los viajeros esperan en fila su turno hasta que esté disponible el trasporte.

Un sistema de bus común puede construirse con multiplexores y un registro de destino para que el bus de trasferencia pueda seleccionarse por medio de un decodificador. Los multiplexores seleccionan un registro fuente para el bus y el decodificador selecciona un registro de destino para trasferir la información desde el bus. La construcción de un sistema de bus para cuatro registros se dibuja en la Figura 8-6. Los cuatro bits en la misma posición significativa de los registros pasan a través de un multiplexor de 4 a 1 línea para formar una línea de bus. Solamente dos multiplexores se muestran en el diagrama: uno para dos bits significativos de menor orden y uno para dos bits significativos de mayor orden. Para re-

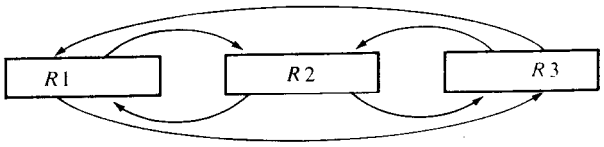


Figura 8-4 Trasferencia entre tres registros

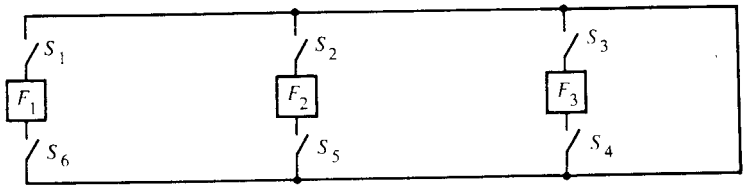


Figura 8-5 Trasferencia a través de una línea común

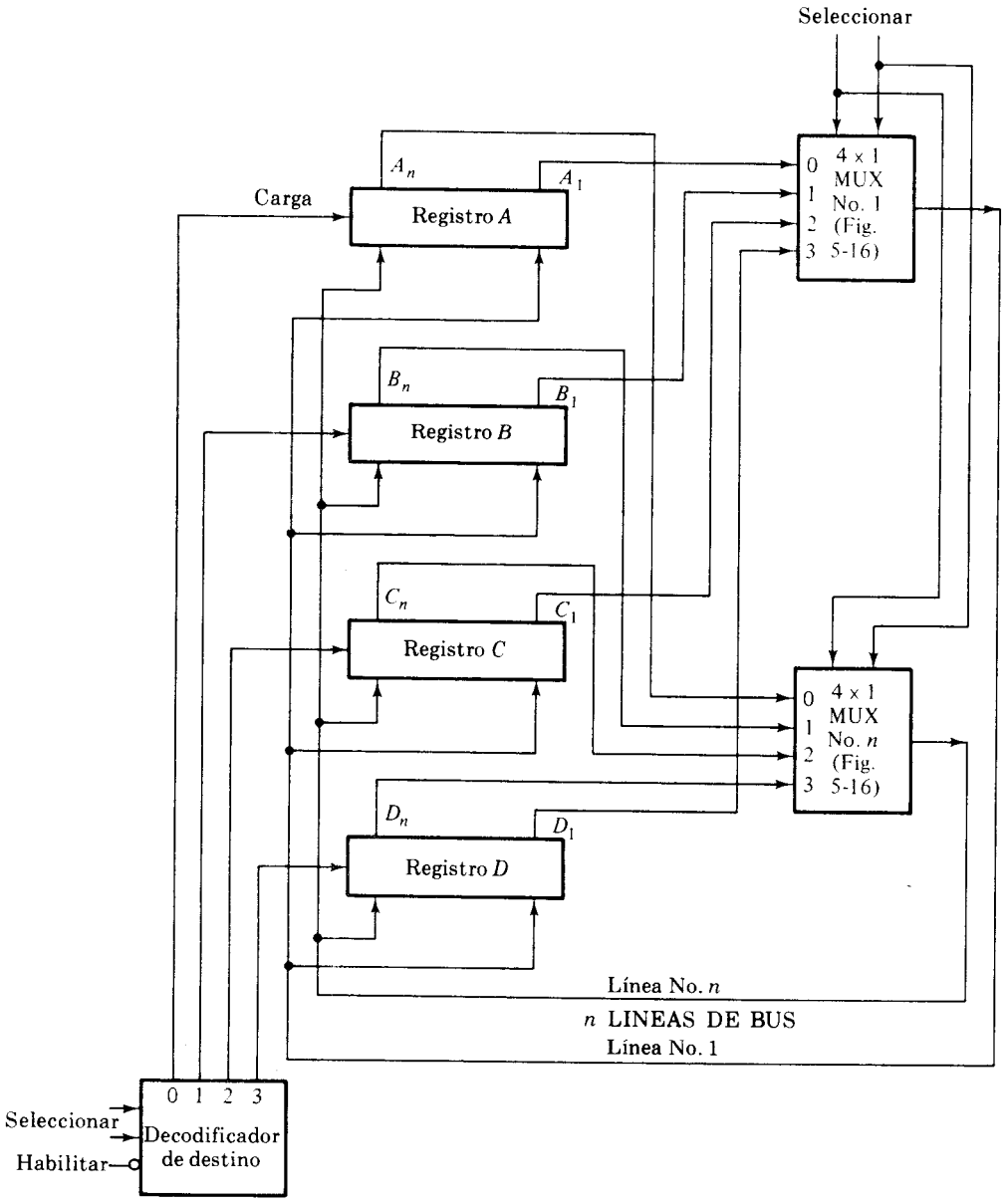


Figura 8-6 Sistema de bus para cuatro registros

gistros de n bits, se necesitan n multiplexores para producir un bus de n líneas. Las n líneas en el bus se conectan a n entradas de todos los registros. La trasferencia de información de un bus a un registro de destino se logra activando el control de carga de ese registro. El control de carga particular activado se selecciona mediante las salidas del decodificador cuando se habilita. Si el decodificador no se habilita, no se trasferirá ninguna información, aunque los multiplexores coloquen el contenido de un registro fuente en el bus.

Para ilustrar lo anterior con un ejemplo particular, considérese la siguiente proposición:

$$C \leftarrow A$$

La función de control que habilita esta trasferencia debe seleccionar el registro A para el bus y el registro C para el destino. Las entradas de selección de los multiplexores y el decodificador deben ser:

Fuente de selección = 00	(los MUX seleccionan los registros A)
Destino seleccionado = 10	(el decodificador selecciona el registro C)
Habilitación decodificador = 0	(el decodificador se habilita)

En el siguiente pulso de reloj el contenido de A, localizado sobre el bus, se carga el registro C.

Trasferencia de memoria

La operación de una unidad de memoria fue descrita, en la Sección 7-7. La trasferencia de información a partir de un registro de memoria al exterior se llama operación de *lectura*. La trasferencia de la información nueva a un registro de memoria se llama la operación de *escritura*. En ambas operaciones, el registro de memoria seleccionado se especifica por medio de una dirección.

Un registro de memoria o palabra se simboliza por medio de la letra M. El registro de memoria particular entre los muchos disponibles en una unidad de memoria se selecciona por medio de la dirección de memoria durante la trasferencia. Es necesario especificar la dirección de M cuando se escriben proposiciones de trasferencias de memorias. En algunas aplicaciones, solamente un registro de direcciones se conecta a los terminales de direcciones de la memoria. En otras aplicaciones, las líneas de dirección forman un sistema de bus común, para permitir que muchos registros especifiquen una dirección. Cuando se conecta solamente un registro a la dirección de memoria, se sabe que este registro especifica la dirección y que se puede adoptar una convención que simplifica la notación. Si la letra M aparece por sí sola en una proposición, designará siempre un registro de memoria seleccionado por la dirección que está al presente en el MAR. De otra manera, el registro que especifica la dirección (o la dirección en sí) se encerrará entre llaves cuadradas después del símbolo M.

Considérese una unidad de memoria que tenga un solo registro de direcciones *MAR* como se muestra en la Figura 8-7. El diagrama muestra también un solo registro separador de memoria *MBR* usado para trasferir datos hacia adentro y afuera de la memoria. Hay dos operaciones de trasferencia de memoria: lectura y escritura. La operación de lectura es una trasferencia de un registro *M* de memoria seleccionado al *MBR*. Esto se designa simbólicamente por medio de la proposición:

$$R: \quad MBR \leftarrow M$$

R es la función de control que inicia la operación de lectura. Esto causa la trasferencia de la información al *MBR* del registro seleccionado de memoria *M* especificado por la dirección en el *MAR*. La operación de escritura es una trasferencia del *MBR* al registro de memoria seleccionado *M*. Esto se designa por medio de la siguiente proposición:

$$W: \quad M \leftarrow MBR$$

W es la función de control que inicia la operación de escritura. Esta última causa una trasferencia de la información del *MBR* al registro de memoria *M* seleccionado por la dirección presente en el *MAR*.

El tiempo de acceso de una unidad de memoria debe estar sincronizado con los pulsos maestros de reloj en el sistema que dispara los registros del procesador. En memorias rápidas el tiempo de acceso debe ser menor que o igual a un período de pulso de reloj. En memorias lentas, podría ser necesario esperar por un número de pulsos de reloj, para que se complete la trasferencia. En memorias de núcleos magnéticos, los registros del procesador deben esperar para que el tiempo de ciclo de memoria se complete. Para una operación de lectura, el tiempo de ciclo incluye la restauración de la palabra después de la lectura. Para una operación de escritura, el tiempo de ciclo incluye el borrado de la palabra de memoria después de la lectura.

En algunos sistemas, la unidad de memoria recibe direcciones y datos de muchos registros conectados a los buses comunes. Considérese el caso dibujado en la Figura 8-8. La dirección a la unidad de memoria viene de un bus de dirección. Se conectan cuatro registros a este bus y cualquiera puede suministrar una dirección. La salida de la memoria puede ir a cualquiera de los cuatro registros, los cuales se seleccionan por medio de un decodificador. La entrada de datos a la memoria viene del bus de datos, la

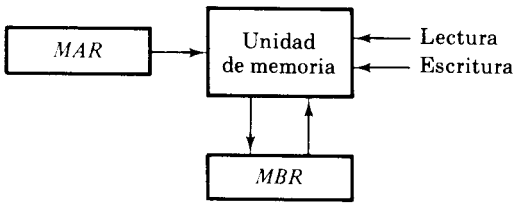


Figura 8-7 Unidad de memoria que se comunica con dos registros externos

cual selecciona uno de los cuatro registros. Una palabra de memoria se especifica en tal sistema por medio del símbolo M seguido por un registro encerrado en llaves cuadradas. El contenido del registro dentro de las llaves cuadradas especifica la dirección de M. La trasferencia de información del registro $B2$ a una palabra seleccionada de memoria por la dirección en el registro $A1$ se simboliza por medio de la proposición:

$$W: \quad M[A1] \leftarrow B2$$

Esta es una operación de escritura, con el registro $A1$ especificando la dirección. Las llaves cuadradas después de la letra M dan el registro direccionado usado para seleccionar el registro de memoria M. La proposición no especifica explícitamente los buses. Empero, ésta implica las entradas de selección requeridas por los dos multiplexores que forman los buses de dirección y de datos.

La operación de lectura en una memoria con buses puede especificarse de manera similar. La proposición:

$$R: \quad B0 \leftarrow M[A3]$$

simboliza una operación de lectura de un registro de memoria cuya dirección está dada por $A3$. La información binaria que sale de la memoria se trasfiere al registro $B0$. De nuevo, esta declaración implica las entradas de selección requeridas por el multiplexor direccionado y las variables de selección para el decodificador de destino.

8-3 MICROOPERACIONES ARITMETICAS, LOGICAS Y DESPLAZAMIENTO

Las microoperaciones de trasferencia entre registros no cambian el contenido de información binaria, cuando ésta pasa del registro fuente al registro de destino. Todas las demás microoperaciones cambian el contenido de la información durante la trasferencia. Entre todas las operaciones posibles que pueden existir en un sistema digital, hay un conjunto básico del cual todas las demás operaciones pueden obtenerse. En esta sección se define un conjunto de microoperaciones básicas, su notación simbólica y los materiales digitales que las configuran. Se pueden definir otras microoperaciones con símbolos adecuados, si es necesario, para amoldarse a una aplicación particular.

Microoperaciones aritméticas

Las microoperaciones aritméticas básicas son: sumar, restar, complementar y desplazar. Los desplazamientos aritméticos se explican en la Sección 8-7 conjuntamente con el tipo de representación en datos binarios. Todas las demás operaciones aritméticas pueden obtenerse de una variación o secuencia de estas microoperaciones básicas.

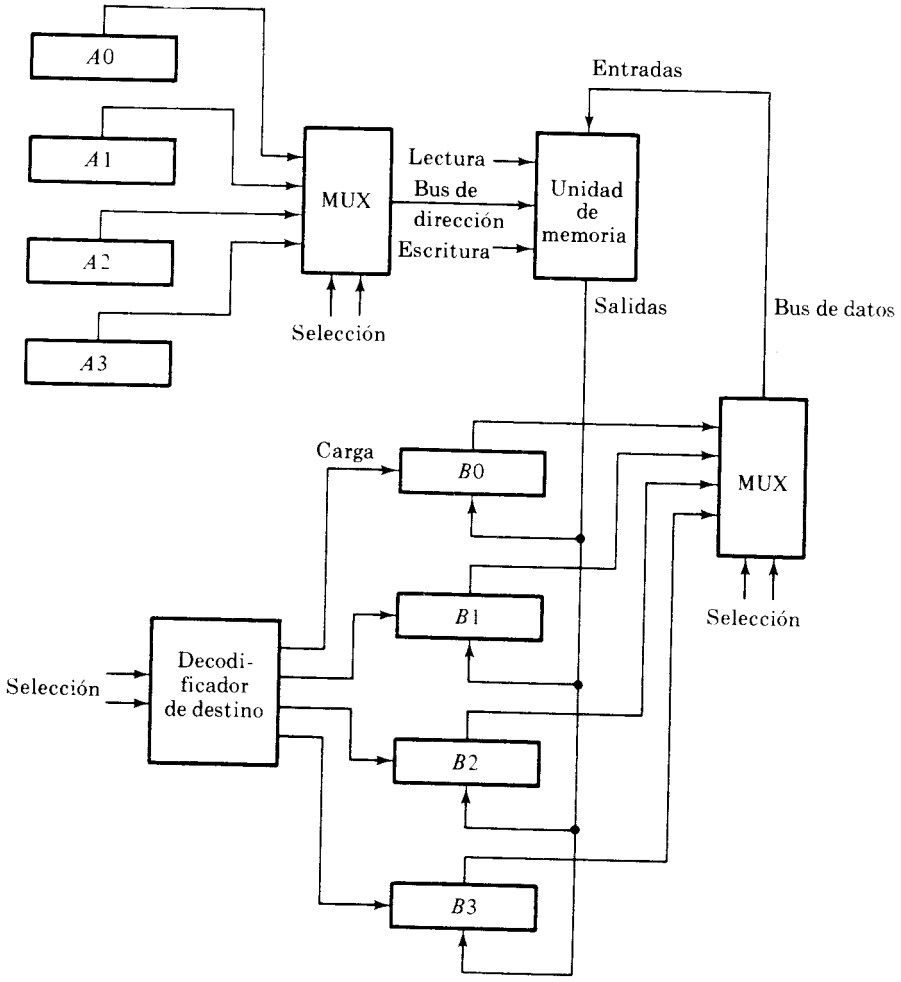


Figura 8-8 Unidad de memoria que se comunica con múltiples registros

La microoperación aritmética se define por la proposición:

$$F \leftarrow A + B$$

la cual especifica una operación de *suma*. Esta establece que el contenido del registro A se va a sumar al contenido del registro B, y la suma se trasfiere al registro F. Para configurar la proposición, se requieren tres registros, A, B y F y la función digital que realiza la operación de suma, tal como un sumador en paralelo. Las otras operaciones aritméticas básicas se listan en la Tabla 8-2. La sustracción aritmética implica la disponibilidad de un sustractor paralelo binario compuesto de circuitos sustractores completos conectados en cascada. La sustracción se configura a menudo

por medio de la complementación y suma como se especifica por la siguiente proposición:

$$F \leftarrow A + \overline{B} + 1$$

$\overline{B}$ es el símbolo para el complemento de 1 de B. Al agregar 1 al complemento de 1, dará el complemento de 2 de B. Agregando A al complemento de 2 de B, se producirá A menos B.

Las microoperaciones de incremento y decremento se simbolizan por una operación de *más uno* ó *menos uno* ejecutadas con los contenidos del registro. Estas microoperaciones se configuran con un contador creciente o decreciente respectivamente.

Debe haber una relación directa entre las proposiciones escritas en un lenguaje de trasferencia entre registros y los registros y funciones digitales que se necesitan para su configuración. Para ilustrar esta relación, considérese las dos proposiciones:

$$T_2: \quad A \leftarrow A + B$$
$$T_5: \quad A \leftarrow A + 1$$

Tabla 8-2 Microoperaciones aritméticas

Designación simbólica	Descripción
$F \leftarrow A + B$	Contenido de A más B se trasfiere a F
$F \leftarrow A - B$	Contenido de A menos B se trasfiere a F
$B \leftarrow \overline{B}$	Se complementa el registro B (complemento de 1)
$B \leftarrow \overline{B} + 1$	Formar el complemento de 2 del contenido del registro B.
$F \leftarrow A + \overline{B} + 1$	A más el complemento de 2 de B se trasfiere a F
$A \leftarrow A + 1$	Incrementar el contenido de A en 1 (cuenta creciente)
$A \leftarrow A - 1$	Decrementar el contenido de A en 1 (cuenta decreciente)

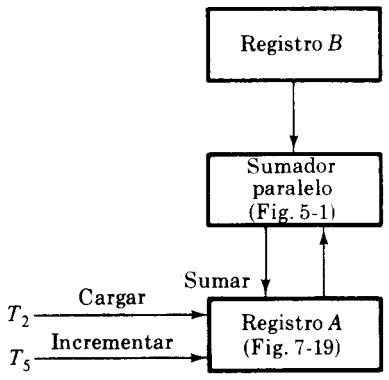


Figura 8-9 Configuración para las microoperaciones de suma e incremento

La variable de tiempo T_2 inicia una operación para sumar el contenido del registro B al contenido presente de A. La variable de tiempo T_5 incrementa el registro A. El incremento puede hacerse fácilmente con un contador y la suma de dos números binarios puede generarse con un sumador en paralelo. La trasferencia de la suma del sumador en paralelo al registro A puede activarse con una entrada de carga al registro. Esto indica que el registro es un contador con capacidad de carga en paralelo. La configuración de las dos declaraciones se muestra en el diagrama de bloque en la Figura 8-9. Un sumador paralelo recibe información de entrada de los registros A y B. Los bits suma del sumador paralelo se aplican a las entradas de A y la variable de tiempo T_2 carga la suma al registro A. La variable de tiempo T_5 incrementa el registro habilitando la entrada de incremento (o entrada de conteo como en la Figura 7-19).

Nótese que las operaciones aritméticas de multiplicación y división no están listadas en la Tabla 8-2. La operación de multiplicación puede ser representada por el símbolo *, y la división por un /. Estas dos operaciones son operaciones aritméticas válidas pero no se incluyen en el conjunto básico de microoperaciones. El único lugar donde estas operaciones pueden considerarse como microoperaciones es un sistema digital en donde se configuran por medio de los circuitos combinacionales. En tal caso, las señales que ejecutan estas operaciones se propagan a través de las compuertas, y los resultados de la operación pueden ser trasferidos a un registro de destino por medio de un pulso de reloj, tan pronto se propagan las señales de salida a través del circuito combinacional. En la mayoría de los computadores, la operación de multiplicación se ejecuta con una secuencia de microoperaciones de suma y desplazamiento. La división se ejecuta con una secuencia de microoperaciones de resta y desplazamiento. Para especificar la configuración de los materiales en tal caso, se requiere una lista de proposiciones que usan microoperaciones básicas de *suma, resta* y *desplazamiento.*

Microoperaciones lógicas

Las microoperaciones lógicas especifican operaciones binarias para una cadena de bits almacenados en los registros. Estas operaciones consideran cada bit en los registros separadamente y lo tratan como una variable binaria. Como ilustración, la microoperación del OR exclusivo se simboliza por medio de la proposición:

$$F \leftarrow A \oplus B$$

Esta especifica una operación lógica que considera cada par de bits en los registros como variables binarias. Si el contenido del registro A es 1010 y el del registro B 1100, la información trasferida al registro F es 0110:

1010	contenido de A
1100	contenido de B
0110	contenido de $F \leftarrow A \oplus B$

Hay 16 operaciones lógicas diferentes posibles que pueden realizarse con dos variables binarias. Estas operaciones lógicas se listan en la Tabla 2-6. Todas las 16 operaciones pueden expresarse en términos de AND, OR y complemento. Se adoptarán símbolos especiales para estas tres microoperaciones para distinguirlas de los símbolos correspondientes usados para expresar funciones de Boole. El símbolo $\vee$ se usará para demostrar una microoperación OR y el símbolo $\wedge$ para denotar una microoperación AND. La microoperación complemento es la misma que el complemento de 1 y usa una barra encima de la letra (o letras) que denotan el registro. Usando estos símbolos, es posible diferenciar entre una microoperación lógica y una función de control (o de Boole). Los símbolos para las cuatro microoperaciones lógicas se sumarizan en la Tabla 8-3. Los últimos dos símbolos son para las microoperaciones de desplazamiento expuestas a continuación.

Tabla 8-3 Microoperaciones lógicas y de desplazamiento

Designación simbólica	Descripción
$A \leftarrow \overline{A}$	Complementa todos los bits del registro A
$F \leftarrow A \vee B$	Microoperación OR lógica
$F \leftarrow A \wedge B$	Microoperación AND lógica
$F \leftarrow A \oplus B$	Microoperación OR exclusiva lógica
$A \leftarrow \text{shl } A$	Registro A de desplazamiento a la izquierda
$A \leftarrow \text{shr } A$	Registro A de desplazamiento a la derecha

Una razón muy importante para adoptar un símbolo especial para la microoperación OR es diferenciar el símbolo + cuando se usa como un más aritmético en una operación lógica OR. Aunque el símbolo + tiene dos significados, es posible distinguirlos notando cuando ocurren los símbolos. Cuando este símbolo se presenta en una microoperación, denota un más aritmético. Cuando ocurre en una función de control (o de Boole) denota una operación lógica OR. Por ejemplo en la declaración:

$$T_1 + T_2\colon \quad A \leftarrow A + B, \quad C \leftarrow D \vee F$$

el + entre T_1 y T_2 es una operación OR entre dos variables de tiempo de una función de control. El + entre A y B especifica una microoperación de suma. La microoperación OR se distingue por el símbolo $\vee$ entre los registros D y F.

Las microoperaciones lógicas pueden configurarse fácilmente con un grupo de compuertas. El complemento de un registro de n bits se obtiene de n compuertas inversoras. La microoperación AND se obtiene de un grupo de compuertas AND, cada una de las cuales recibe un par de bits de los dos registros fuente. Las salidas de las compuertas AND se aplican a las entradas del registro de destino. La microoperación OR requiere un grupo de compuertas OR dispuestas de manera similar.

Microoperaciones de desplazamiento

Las microoperaciones de desplazamiento trasfieren la información binaria entre registros en los computadores en serie. Se usan también en computadores en paralelo para operaciones aritméticas, lógicas y de control. Los registros pueden trasferirse a la izquierda o a la derecha. No hay símbolos convencionales para las operaciones de desplazamiento. En este libro, se adoptan los símbolos convencionales para las operaciones de desplazamiento. En este libro, se adoptan los símbolos *shl* y *shr* para las operaciones de desplazamiento a la izquierda y a la derecha respectivamente. Por ejemplo:

$$A \leftarrow \text{shl } A, \qquad B \leftarrow \text{shr } B$$

son dos microoperaciones que especifican un desplazamiento de 1 bit a la izquierda del registro A y 1 bit a la derecha del registro B. El símbolo de registro debe ser el mismo en ambos lados de la flecha como una operación de incremento.

Mientras los bits de un registro se desplazan, los flip-flops extremos reciben información de la entrada en serie. El flip-flop extremo está en la posición de extrema izquierda del registro, durante una operación de desplazamiento a la derecha y en la posición de extrema izquierda durante una operación de desplazamiento a la izquierda. La información trasferida a los flip-flops extremos no se especifica por los símbolos *shl* y *shr*. Por tanto, una proposición de una microoperación de desplazamiento debe estar acompañada con otra microoperación que especifica el valor de la entrada en serie del bit trasferido al flip-flop extremo. Por ejemplo:

$$A \leftarrow \text{shl } A, \qquad A_1 \leftarrow A_n$$

es un desplazamiento circular que trasfiere el bit de la extrema izquierda desde A_n hasta el flip-flop de la extrema derecha A_1. De manera similar:

$$B \leftarrow \text{shr } B, \qquad A_n \leftarrow E$$

es una operación de desplazamiento a la derecha con el flip-flop de la extrema izquierda A_n recibiendo el valor del registro E de 1 bit.

8-4 PROPOSICIONES CONDICIONALES DE CONTROL

Es conveniente algunas veces especificar una condición de control por medio de una proposición condicional en vez de una función de control de Boole. Una proposición de *control condicional* se simboliza por medio de una proposición de *si-entonces-por tanto* de la siguiente manera:

P: *si* (condición) *entonces* [microoperación(es)]
por tanto [microoperación(es)]

La proposición se interpreta de manera que si la condición de control, establecida entre paréntesis después de la palabra *si*, es verdadera, enton-

ces se ejecuta la microoperación (o microoperaciones) encerrada entre paréntesis después de la palabra *entonces*. Si la condición no es verdadera, se ejecuta la microoperación listada después de la palabra *por tanto*. De cualquier forma, la función de control P debe ocurrir para cualquier evento que se haga. Si la parte *por tanto* falta, entonces si la condición no es verdadera no se ejecuta nada.

La proposición de control condicional es más una conveniencia que una necesidad. Esta habilita la escritura de proposiciones mas claras que son más fáciles de interpretar por la gente. Puede ser reescrita por una proposición convencional sin la forma si-entonces-por tanto.

Como ejemplo, considérese la proposición de control condicional:

$$T_2\text{: si}(C = 0) \text{ entonces } (F \leftarrow 1) \text{ por tanto } (F \leftarrow 0)$$

Se asume que F es un registro de 1 bit (flip-flop) que puede ser puesto a 1 o borrado. Si el registro C es un registro de 1 bit, la afirmacion es equivalente a las dos proposiciones siguientes:

$$C'T_2\text{: } F \leftarrow 1$$
$$CT_2\text{: } F \leftarrow 0$$

Nótese que la misma variable de tiempo puede ocurrir en dos funciones de control separadas. La variable C puede ser 0 ó 1; por tanto solamente una de las microoperaciones se ejecutan durante T_2, dependiendo del valor de C.

Si el registro C tiene más de un bit, la condición $C = 0$ significa que todos los bits de C deben ser 0. Al asumir que el registro C tiene cuatro bits C_1, C_2, C_3 y C_4 la condición para $C = 0$ puede ser expresada con una función de Boole:

$$x = C_1'C_2'C_3'C_4' = (C_1 + C_2 + C_3 + C_4)'$$

La variable x puede ser generada con una compuerta NOR. Usando la definición de x como se estableció, la proposición del control condicional es equivalente a dos proposiciones:

$$xT_2\text{: } F \leftarrow 1$$
$$x'T_2\text{: } F \leftarrow 0$$

La variable $x = 1$ si $C = 0$ pero es igual a 0 si $C \neq 0$.

Cuando se escriben proposiciones de control condicional, se debe tener en cuenta que la proposición establecida después de la palabra *si*, es parte de la función de control y no parte de la proposición de microoperación. La condición debe establecerse claramente y debe poder configurarse con un circuito combinacional.

8-5 DATOS BINARIOS DEL PUNTO FIJO

La información binaria encontrada en los registros representa datos o información de control. Los datos son operandos y otros elementos discretos de información con los cuales se opera para lograr los resultados requeridos. La información de control es un bit o grupo de bits que especifican las operaciones que se van a ejecutar. Una unidad de información de control almacenada en los registros de computador digital se llama *instrucción* y es un código binario que especifica las operaciones que se van a realizar con los datos acumulados. Los códigos de instrucción y su representación en los registros se presentan en la Sección 8-11. Al final de las siguientes secciones se presentan algunos tipos comunes de datos y su representación.

Representación del signo y el punto radical

Un registro con n flip-flops puede almacenar un número binario de n bits; cada flip-flop representa un dígito binario. Este representa la magnitud del número pero no da información acerca de su signo o la posición del punto binario. El signo se necesita para operaciones aritméticas ya que representa cuando el número es positivo o negativo. La posición del punto decimal es necesaria para representar enteros, fracciones o números enteros y fraccionarios mezclados.

El signo de un número es una cantidad discreta de información que tiene dos valores: más o menos. Estos dos valores pueden ser representados por un código de un bit. La convención es representar un más con un 0 y un menos con un 1. Para representar un número binario con signo, se necesitan $n = k + 1$ flip-flops, k de ellos para la magnitud y uno para almacenar el signo del número.

La representación del punto binario se complica por el hecho de que éste se caracteriza por una *posición* entre los dos flip-flops en el registro. Hay dos maneras posibles de especificar la posición del punto binario en un registro: dándole una posición de *punto fijo* o empleando una representación de *punto flotante*. El método del punto fijo asume que el punto binario está siempre fijo en un posición. Las dos posiciones más usadas son (1) un punto binario en el extremo izquierdo del registro para hacer del numero almacenado una fracción, y (2) un punto binario en el extremo del registro para hacer del número almacenado un entero. En ambos casos el punto binario no es visible físicamente, pero se asume a partir del hecho de que el número almacenado en el registro se trata como una fracción o como un entero. La representación del punto flotante usa un segundo registro para almacenar un número que designa la posición del punto binario en el primer registro. La representación del punto flotante se explica en la Sección 8-9.

Números binarios con signos

Cuando un número binario de punto fijo es positivo, el signo se representa como 0 y la magnitud por un número binario positivo. Cuando el número es

negativo, el signo se representa por un 1 y el resto del número puede ser representado por cualquiera de las tres maneras siguientes. Estas son:

1. Signo-magnitud.
2. Signo-complemento de 1.
3. Signo-complemento de 2.

En la representación de la magnitud del signo, ésta se representa por un número binario positivo. En las otras dos representaciones, el número estará en complemento de 2 ó de 1. Si el número es positivo, las tres representaciones son iguales.

Como ejemplo, el número binario 9 se escribe a continuación en tres modalidades. Se asume que se dispone de un registro de 7 bits para almacenar el signo y la magnitud del número.

	+9	−9
Signo-magnitud	0 001001	1 001001
Signo-complemento de 1	0 001001	1 110110
Signo-complemento de 2	0 001001	1 110111

Un número positivo en cualquier representación tiene un 0 en el bit de la extrema izquierda para un más, seguido de un número binario positivo. Un número negativo siempre tiene un 1 en el bit de la extrema izquierda para un menos, pero los bits de magnitud se representan en forma diferente. En la representación de signo-magnitud, estos bits son el número positivo; en la representación del complemento de 1, estos bits son el complemento del número binario; y en la representación del complemento de 2, el número está en su forma de complemento de 2.

La clara representación del signo-magnitud de -9 se obtiene de $+9$ (0 001001) complementando *solamente* el bit del signo. La representación de signo-complemento de 1 de -9 se obtiene complementando *todos* los bits de 0 001001 ($+9$), incluyendo el bit del signo. La representación de signo-complemento de 2 se logra obteniendo el complemento de 2 del número positivo, *incluyendo* su bit de signo.

Suma aritmética

La razón para usar la representación de signo-complemento para los números negativos se hará aparente una vez se consideren los diferentes pasos para formar la suma de dos números con signo. La representación de signo-magnitud es la que más se usa en los cálculos cotidianos. Por ejemplo, $+23$ y -35 son representados con un signo seguido por la magnitud del número. Para sumar estas dos funciones, es necesario restar la magnitud menor de la magnitud mayor y usar el signo del número mayor como el signo del resultado, es decir $(+23) + (-35) = -(35-23) = -12$. El

proceso de sumar dos números con signo, cuando los números negativos están representados en la forma de signo-magnitud, requiere que se comparen estos signos. Si los dos signos no son iguales, se comparan las magnitudes relativas de los números y luego se resta el menor del mayor. Es necesario determinar también el signo del resultado. Este es un proceso que requiere una secuencia de decisiones de control de la misma que circuitos que puedan comparar, sumar y restar números, cuando se configura con materiales digitales.

Compárese ahora el procedimiento anterior con el procedimiento que forma la suma de dos números binarios con signo, cuando los números negativos están representados en la forma de complemento de 1 ó 2. Estos procedimientos son muy simples y pueden formularse de la siguiente manera:

Suma representada por signo-complemento de 2. La suma de dos números binarios con signo y los números negativos representados por sus complementos de 2 se obtienen de la suma de dos números con sus bits de signo incluidos. Se descarta el arrastre en el bit más significativo (signo).

Suma representada por signo-complemento de 1. La suma de dos números binarios con números negativos representados por sus complementos de 1, se obtienen de la suma de dos números, con sus bits de signo incluidos. Si hay un arrastre del bit más significativo (signo), el resultado se incrementa en 1 y el arrastre se descarta.

Los ejemplos numéricos para la suma con números negativos, representados por su complemento de 2, se muestran a continuación. Nótese que dos números negativos deben estar inicialmente *representados por su complemento de 2* y que la suma obtenida después de la adición estará siempre con la representación adecuada.

+ 6	0 000110	− 6	1 111010
+		+	
+ 9	0 001001	+ 9	0 001001
+ 15	0 001111	+ 3	0 000011
+ 6	0 000110	− 9	1 110111
+		+	
− 9	1 110111	− 9	1 110111
− 3	1 111101	− 18	1 101110

Los dos números de los cuatro ejemplos se suman, con sus bits de signo incluidos. Cualquier arrastre del bit de signo se descarta y los resul-

tados negativos se producen automáticamente en la forma de complemento de 2.

Los cuatro ejemplos se repiten a continuación con los números negativos representados por su complemento de 1. El arrastre del bit de signo se regresa y agrega al bit menos significativo (arrastre final lleva final de reinicio).

+ 6	0 000110		− 6	1 111001	
		+			+
+ 9	0 001001		+ 9	0 001001	
+ 15	0 001111			10 000010	
					+
				→ 1	
			+ 3	0 000011	
+ 6	0 000110		− 9	1 110110	
		+			+
− 9	1 110110		− 9	1 110110	
− 3	1 111100			11 101100	
					+
				→ 1	
			− 18	1 101101	

La ventaja de la representación en la forma de signo-complemento de 2 sobre la forma signo-complemento de 1 (y la forma signo-magnitud) es que la primera contiene solamente un tipo de cero. Las otras dos representaciones tienen ambas un cero positivo y un cero negativo. Por ejemplo, agregándo +9 a −9 en la representación de complemento de 1, se obtiene:

+9	0 001001
−9	1 110110
−0	1 111111

y el resultado es un cero negativo, es decir, el complemento de 0 000000 (cero positivo).

Un cero con su bit de signo asociado aparecerá en el registro en una de las siguientes formas dependiendo de la representación usada para números negativos:

	+0	−0
En signo-magnitud	0 0000000	1 0000000
En signo-complemento de 1	0 0000000	1 1111111
En signo-complemento de 2	0 0000000	ninguna

Ambas representaciones de signo-magnitud y complemento de 1 tienen asociadas con ellas la posibilidad de un cero negativo. La representación del signo-complemento de 2 tiene solamente un cero positivo. Esto ocurre debido a que el complemento de 2 de 0 000000 (cero positivo) es 0 000000 y puede ser obtenido del complemento de 1 más 1 (es decir 1 111111 + 1) teniendo en cuenta que se descarta el arrastre final o lleva final de reinicio.

El rango de los números enteros binarios que pueden ser acomodados en un registro de $n = k + 1$ bit es $\pm(2^k - 1)$, donde se reservan k bits para el número y un bit para el signo. Un registro con 8 bits puede almacenar números binarios en el rango de $\pm(2^7 - 1) = \pm 127$. Sin embargo, como la representación de signo complemento de 2 tiene solamente un cero, debe acomodar un número más que las otras dos representaciones. Considérese la representación de los números mayores y menores:

	Signo complemento de 1	Signo complemento de 2
+126 = 0 1111110	− 126 = 1 0000001	1 0000010
+127 = 0 1111111	−127 = 1 0000000	1 0000001
+128 (imposible)	− 128 (imposible)	1 0000000

En la representación de signo-complemento de 2, es posible representar − 128 con ocho bits. En general, la representación de signo-complemento de 2 puede acomodar números en el rango $+(2^k - 1)$ a -2^k, donde $k = n - 1$ y n es el número de bits en el registro.

Sustracción aritmética

La sustracción de dos números binarios con signo, cuando los números negativos están en la forma de complemento de 2, es muy simple y puede exponerse como sigue: *Obténgase el complemento de 2 del sustraendo (incluyendo el signo de bit) y súmese al minuendo (incluyendo el bit de signo).* Este procedimiento hace uso del hecho de que una operación de resta puede cambiarse a una operación de suma si el signo del sustraendo se cambia. Esto se demuestra por las siguientes relaciones (B es el sustraendo):

$$(\pm A) - (-B) = (\pm A) + (+B)$$
$$(\pm A) - (+B) = (\pm A) + (-B)$$

Cambiar un número positivo a un número negativo se hace fácilmente tomando el complemento de 2 (incluyendo el bit de signo). Lo contrario es también verdad, porque el complemento del complemento regresa al número a su valor original.

La sustracción con números en complemento de 1 es similar, excepto por el arrastre final o lleva final de reinicio. La sustracción con signo-magnitud requiere que solamente el bit signo del sustraendo se complemente. La suma y resta de los números binarios en la representación de signo-magnitud se demuestra en la Sección 10-3.

Debido a que el procedimiento más sencillo para sumar y restar números binarios con números negativos lo constituye la forma de signo-complemento de 2, la mayoría de las computadoras adoptan esta representación sobre la forma más familiar de signo-magnitud. La razón por la cual el complemento de 2 se escoge, en vez del complemento de 1, es para evitar el arrastre final o lleva final de reinicio y la ocurrencia de un cero negativo.

8-6 SOBRECAPACIDAD

Cuando dos números con n dígitos cada uno se suman y la suma ocupa $n+1$ dígitos, se dice que hay un desbordamiento por *sobrecapacidad*. Esto es verdadero para los números binarios o números decimales con o sin signo. Cuando se hace una suma con lápiz y papel, una sobrecapacidad no es un problema ya que no hay limitaciones por el ancho de la página para escribir la suma. Una sobrecapacidad es un problema en un computador digital ya que las longitudes de todos los registros, incluyendo todos los registros de memoria son de longitud finita. Un resultado de $n+1$ bits no puede acomodarse en un registro de longitud normalizada n. Por esta razón, muchos computadores comprueban la ocurrencia de la sobrecapacidad y cuando esto ocurre, ponen a 1 el flip-flop de sobrecapacidad para que el usuario verifique.

Un sobrecapacidad no puede ocurrir después de una suma si un número es positivo y el otro es negativo ya que agregando un número positivo a un número negativo produce un resultado (positivo o negativo), el cual es menor que el mayor de los dos números originales. Una sobrecapacidad puede ocurrir si los dos números se suman y ambos son positivos o ambos negativos. Cuando se suman dos números representados en signo-magnitud, se puede detectar fácilmente una sobrecapacidad por el arrastre o el número de bits. Cuando se suman dos números representados en signo-complemento de 2, el bit signo se trata como parte del número pero no necesariamente indica una sobrecapacidad.

El algoritmo para sumar dos números representados por signo-complemento de 2, como se ha establecido antes, produce un resultado incorrecto cuando sucede una sobrecapacidad. Esto ocurre debido a que una sobrecapacidad de los bits del número cambian siempre el signo del resultado y se causa una respuesta errónea de n bits. Para observar cómo ocurre esto, considérese el siguiente ejemplo: dos números binarios con signo 35 y 40 se almacenan en dos registros de 7 bits. La capacidad máxima del registro es $(2^6 - 1) = 63$ y la capacidad mínima es $-6^6 = -64$. Como la suma de los números es 75, esta excede la capacidad del registro. Esto es valedero si los números son ambos positivos o negativos. Las operaciones en binarios se muestran a continuación conjuntamente con los dos últimos arrastres de la suma:

arrastre:	0	1	arrastre:	1	0
+35	0	100011	−35	1	011101
+40	0	101000	−40	1	011000
−75	1	001011	−75	0	110101

En ambos casos, se observa que el resultado de 7 bits, que debería ser positivo, es negativo o viceversa. Obviamente, la respuesta binaria es incorrecta y el algoritmo para sumar números binarios representados en la forma de complemento de 2, como se ha establecido antes, falla en producir resultados correctos cuando ocurre una sobrecapacidad. Nótese que si el arrastre que se emana de la posición del bit de signo se toma como el signo del resultado, entonces los 8 bits de la respuesta serán correctos.

Una condición de sobrecapacidad puede ser detectada observando el arrastre *a la* posición del bit del signo y el arrastre *de la* posición del bit del signo. Si estas dos categorías no son iguales, se producen condiciones de sobrecapacidad. Esto se indica en el ejemplo anterior en el cual se muestran explícitamente las dos categorías. El lector puede tratar varios ejemplos de números que no producen una sobrecapacidad para observar que estos dos arrastres se convertirán ambos en 0 ó 1. Si estos se aplican a una compuerta OR exclusiva, se detectará una sobrecapacidad cuando la salida de la compuerta es 1.

La suma de dos números binarios con signo, cuando se representan los numeros negativos en la forma de signo y complemento de 2, se configura con funciones digitales como se muestra en la Figura 8-10. El registro A almacena un sumando con su bit de signo en la posición A_n. El registro B almacena el otro sumando con su bit de signo en B_n. Los dos números se suman por medio de un sumador en paralelo de n bits. El circuito sumador completo (FA) en la etapa n (los bits de signo) se muestra explícitamente. El arrastre que va a este sumador completo es C_n. El arrastre

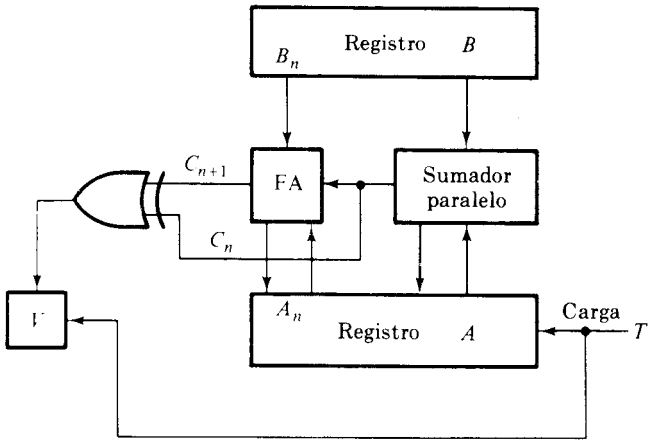


Figura 8-10 Suma de números en signo-complemento de 2

que sale del sumador es C_{n+1}. La función OR exclusiva de estos dos arrastres se aplica a un flip-flop de sobrecapacidad V. Si despues de la suma, $V = 0$, entonces la suma cargada en A es correcta. Si $V = 1$, hay una sobrecapacidad y la suma de n bits es incorrecta. El circuito mostrado en la Figura 8-10 puede especificarse por medio de la siguiente proposicion:

$$T: \quad A \leftarrow A + B, \quad V \leftarrow C_n \oplus C_{n+1}$$

Las variables de la declaración se definen en la Figura 8-10. Notese que las variables C_n y C_{n+1} no representan registros, ellas representan arrastres del sumador paralelo.

8-7 DESPLAZAMIENTOS ARITMETICOS

Un desplazamiento aritmético es una microoperación que mueve un número binario con signo a la izquierda o a la derecha. Un movimiento aritmético a la izquierda multiplica un número binario con signo por 2. Un movimiento aritmético a la izquierda divide el número por 2. Los desplazamientos aritméticos deben dejar el signo sin cambio alguno ya que el signo del número permanece igual cuando se multiplica o divide por 2.

El bit de la extrema izquierda de un registro almacena el bit del signo y los bits restantes almacenan el número. La Figura 8-11 muestra un registro de n bits. El bit A_n de la extrema ezquierda mantiene el bit del signo y se designa como $A(S)$. Los bits del número se almacenan en la parte del registro designada por $A(N)$. A_1 se refiere al bit menos significativo, A_{n-1} se refiere a la posición más significativa de los bits del número, y A se refiere al registro entero.

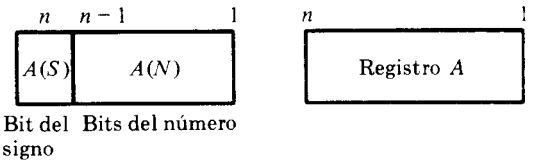


Figura 8-11 Registro que define A para desplazamientos aritméticos

Los números binarios de punto fijo pueden ser representados de tres maneras diferentes. La manera de desplazar el número almacenado en un registro es diferente para cada representación.

Considérese primero un desplazamiento aritmético a la derecha que divide el número por 2. Este puede simbolizarse por cualquiera de las siguientes proposiciones:

$$\left.\begin{array}{l} A(N) \leftarrow \text{shr } A(N), \quad A_{n-1} \leftarrow 0 \text{ para signo-magnitud} \\ A \leftarrow \text{shr } A, \quad A(S) \leftarrow A(S) \text{ para signo-complemento de 1 o signo-complemento 2} \end{array}\right\}$$

En la representación de signo-magnitud, el desplazamiento aritmético a la derecha requiere un movimiento de los bits del número con un 0 colocado

en la posición más significativa. El bit del signo no se afecta. En la representación de signo-complemento de 2 ó de 1, todo el registro se desplaza mientras que el bit del signo permanece inalterado. Esto se debe a que para un número positivo se debe colocar un 0 en la posición más significativa y para un número negativo se debe colocar un 1. Los siguientes ejemplos numéricos ilustran el procedimiento.

Número positivo	+12:	0 01100	+6:	0 00110
Signo-magnitud	−12:	1 01100	−6:	1 00110
Signo-complemento de 1	−12:	1 10011	−6:	1 11001
Signo-complemento de 2	−12:	1 10100	−6:	1 11010

En cada caso el desplazamiento aritmético a la derecha del 12 produce un 6 sin alterar el signo. Para números positivos, el resultado es el mismo en todas las tres representaciones. Un número en signo-magnitud, positivo, negativo, o cuando es desplazado, recibe un 0 en la posición más significativa. La posición más significativa recibe el bit del signo en las dos representaciones de signo-complemento. El último caso es llamado algunas veces *desplazamiento con extensión de signo.*

Considérese ahora el desplazamiento aritmético a la izquierda que multiplica el número por 2. Este puede simbolizarse por cualquiera de las siguientes proposiciones:

$A(N) \leftarrow \text{shl}\ A(N),\quad A_1 \leftarrow 0$	para signo-magnitud
$A \leftarrow \text{shl}\ A,\quad A_1 \leftarrow A(S)$	para signo-complemento de 1
$A \leftarrow \text{shl}\ A,\quad A_1 \leftarrow 0$	para signo-complemento de 2

En la representación de signo magnitud, los bits del número se desplazan a la izquierda con un 0 colocado en la posición menos significativa. En la de signo-complemento de 1 todo el registro se desplaza y el bit del signo se coloca en la posición menos significativa. El signo-complemento de 2 es similar, excepto que un 0 es desplazado a la posición menos significativa. Considérese el número 12 desplazado a la izquierda para producir 24:

Número positivo	0 01100	0 11000
Signo-magnitud	1 01100	1 11000
Signo-complemento de 1	1 10011	1 00111
Signo-complemento de 2	1 10100	1 01000

Un número desplazado a la izquierda puede causar que ocurra un desbordamiento por sobrecapacidad. Una sobrecapacidad ocurrirá después del desplazamiento si existe la siguiente condición *antes* del desplazamiento:

$A_{n-1} = 1$	para signo-magnitud
$A_n \oplus A_{n-1} = 1$	para signo-complemento de 1 o signo-complemento de 2:

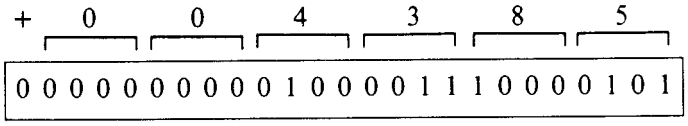
En el caso de signo magnitud, se desplaza y desaparece un 1 de la posición más significativa. En el caso de signo-complemento, ocurrirá la sobrecapacidad si el bit de signo $A_n = A(S)$, no es igual al bit más significativo. Considérese el siguiente ejemplo numérico con números de signo-complemento de 2:

Valor inicial	9:	0 1001	Valor inicial	−9:	1 0110
desplazamiento a la izquierda	−2:	1 0010	desplazamiento a la izquierda	+2:	0 1101

El desplazamiento a la izquierda debería producir 18, pero como el signo original se pierde, se obtiene un resultado incorrecto con una inversión de signo. Si el bit de signo después del desplazamiento no es el mismo que el bit de signo después de él, ocurrirá una sobrecapacidad. El resultado correcto será un número de $n + 1$ bits, con el bit de la posición $(n + 1)$ conteniendo el signo original del número el cual desapareció después del desplazamiento.

8-8 DATOS DECIMALES

La representación de números decimales en los registros es una función del código binario usado para representar un dígito decimal. Un código decimal de 4 bits, por ejemplo, requiere cuatro flip-flops para cada dígito decimal. La representación de +4385 en BCD requiere al menos 17 flip-flops: un flip-flop para el signo y cuatro para cada dígito. Este número se representa en un registro con 25 flip-flops de la siguiente manera:



Al representar los números en decimal, se desperdicia una cantidad considerable de espacio de almacenamiento, ya que el número de flip-flops necesarios para almacenar un número decimal en código binario es mayor que el número de flip-flops necesarios para su representación binaria equivalente. También, los circuitos requeridos para realizar aritmética decimal, son mucho más complejos. Sin embargo, hay algunas ventajas en el uso de la representación decimal, principalmente porque los datos de entrada y salida del computador son generados por personas que siempre usan el sistema decimal. Un computador que usa representación binaria para operaciones aritméticas, requiere conversión de datos de decimal a binario antes de realizar cálculos. Los resultados binarios se deben convertir de nuevo a decimales para la salida. Este procedimiento consume tiempo; vale la pena usarlo en la situación en que las operaciones aritméticas sean enormes, como en el caso de aplicaciones científicas. Algunas aplicaciones, tales como procesamiento de datos de negocios, requieren pequeñas cantidades de cálculos aritméticos. Por esta razón, algunas computadoras

realizan cálculos aritméticos directamente con datos decimales (en código binario) para así eliminar la necesidad de conversión a binario y de nuevo a decimal. Los sistemas de computadores de gran escala comúnmente tienen componentes para realizar cálculos aritméticos en representación binaria y decimal. El usuario puede especificar mediante instrucciones programadas, si el computador va a realizar cálculos en datos binarios o decimales. Un sumador decimal se introdujo en la Sección 5-3.

Hay tres maneras de representar números decimales negativos de punto fijo. Ellas son similares a las tres representaciones de un número binario negativo, excepto por el cambio del radical:

1. Signo-magnitud.
2. Signo-complemento de 9.
3. Signo-complemento de 10.

Un número decimal positivo se representa por un 0 (para el más) seguido por la magnitud del número para todas las tres representaciones. Es con respecto a los números negativos que difieren las representaciones. El signo de un número negativo se representa por un 1 y la magnitud del número es positiva en la representación de signo-magnitud. En las otras dos representaciones la magnitud se representa por el complemento de 9 y de 10.

El signo de un número decimal se toma algunas veces como una cantidad de 4 bits para estar acorde con la representación de 4 bits de los dígitos. Es costumbre representar un más con cuatro ceros y un menos con el equivalente BDC de 9, es decir, 1001. En esta forma todos los procedimientos desarrollados por los números de signo-complemento de 2 se aplican también a los números de signo-complemento de 10. La suma se hace agregando todos los dígitos incluyendo el dígito del signo y descartando el arrastre final o lleva final de reinicio. Por ejemplo, $+375+(-240)$ se hace con la representación de signo-complemento de 10 de la siguiente manera:

$$\begin{array}{rr} & 0\ 375 \\ + & \\ & 9\ 760 \\ \hline & 0\ 135 \end{array}$$

El 9 en el segundo número representa un menos y 760 es el complemento de 10 de 240. Se detecta una sobrecapacidad en esta representación a partir del OR exclusiva de los arrastres que entran y salen de la posición de los dígitos del signo.

Las operaciones aritméticas decimales pueden usar los mismos símbolos que las operaciones binarias siempre y cuando la base de los números se entienda como 10 en vez de 2. La proposición:

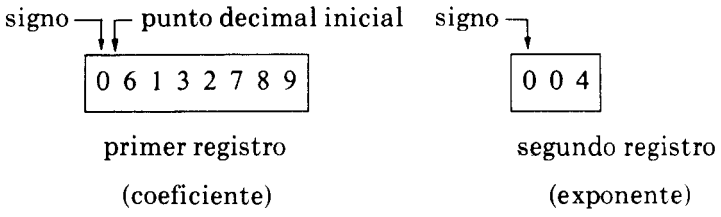
$$A \leftarrow A + \bar{B} + 1$$

puede usarse para expresar la adición del número decimal almacenado en el registro A con el complemento de 10 del número decimal en el registro $\overline{B}$. B en este caso denota el complemento de 9 del número decimal. Los desplazamientos aritméticos son aplicables también a los números decimales excepto que un desplazamiento a la izquierda corresponde a la multiplicación por 10 y un desplazamiento a la derecha a una división por 10. El signo-complemento de 9 es similar al signo complemento de 1 y la representación signo-magnitud en ambas representaciones de radicales tienen procedimientos aritméticos similares.

Si la adopción de símbolos similares para las operaciones binarias y decimales no fueran aceptables, sería necesario formular símbolos diferentes para las operaciones con datos decimales. Algunas veces, las operaciones de registro-trasferencia se usan para simular el sistema por medio de un programa de computador. En tal caso los dos tipos de datos pueden especificarse por declaraciones como se hace en los lenguajes de programación.

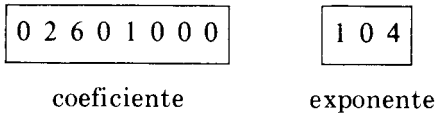
8-9 DATOS DEL PUNTO-FLOTANTE

La representación del punto flotante de los números necesita dos registros. El primero representa un número con signo de punto fijo y el segundo la posición del punto del radical. Por ejemplo, la representación del número decimal $+6132.789$ es de la siguiente manera:

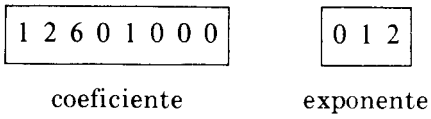


El primer registro tiene un 0 en la posición del flip-flop más significativo para denotar un más. La magnitud del número se almacena en un código binario de 28 flip-flops, con cada dígito decimal ocupando 4 flip-flops. El número en el primer registro se considera una fracción, de manera que el punto decimal en el primer registro se fija a la izquierda del bit más significativo. El segundo registro contiene el número decimal $+4$ (en código binario) para indicar que la posición *actual* del punto decimal es cuatro posiciones decimales a la izquierda. Esta representación es equivalente al número expresado como una fracción multiplicada por 10 a una potencia dada, es decir, $+6132.789$ se representa como $+.6132789 \times 10^{+4}$. Debido a esta analogía, el contenido del primer registro se llama *coeficiente* (y algunas veces *mantisa* o *parte fraccionaria*) y el contenido del segundo registro se llama *exponente* (o *característica*).

La posición del punto decimal actual, puede estar por fuera del rango de los dígitos del registro del coeficiente. Por ejemplo, asumiendo una representación de signo-magnitud, el siguiente contenido:



representa el número $+.2601000 \times 10^{-4} = +.000026010000$, los cuales producen cuatro ceros de más a la izquierda. Por otra parte, el siguiente contenido:

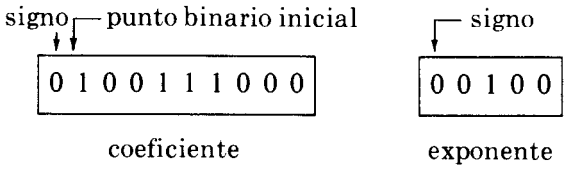


representa el número $-.2601000 \times 10^{12} = -260100000000$, lo cual produce cinco ceros de más a la derecha.

En estos ejemplos, se asume que el coeficiente es una fracción de punto fijo. Algunos computadores lo asumen como un entero, de manera que el punto decimal inicial en el registro del coeficiente está a la derecha del dígito menos significativo.

Otra disposición usada para el exponente es quitar del todo su bit de signo y considerar el exponente como "polarizado". Por ejemplo los números entre 10^{+49} y 10^{-50} pueden representarse con un exponente de dos dígitos (sin el bit de signo) y una polarización de 50. El registro del exponente siempre contiene el número $E + 50$, E es el exponente actual. La sustracción de 50 del contenido del registro dará el exponente deseado. En esta forma, los exponentes positivos se representan en el registro en el rango de números entre 50 a 99. La sustracción de 50 dará los valores positivos desde 00 hasta 49. Los exponentes negativos se representan en el registro en el rango de 00 hasta 49. La sustracción de 50 da los valores negativos en el rango de -50 a -1.

Un número binario de punto flotante se representa de manera similar con dos registros, uno para almacenar el coeficiente y el otro para el exponente. Por ejemplo el número $+1001.110$ puede representarse de la siguiente manera:

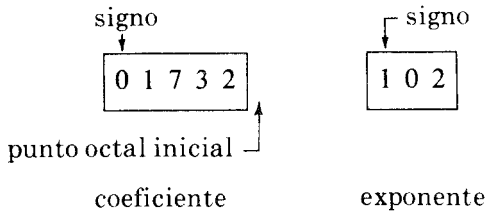


El registro del coeficiente tiene 10 flip-flops: una para el signo y nueve para la magnitud. Asumiendo que el coeficiente es una fracción de punto fijo, el punto binario actual es cuatro posiciones a la derecha, de manera que el exponente tiene el valor binario $+4$. El número se representa en binario como $.100111000 \times 10^{100}$ (recuérdese que 10^{100} en binario es equivalente al decimal 2^4).

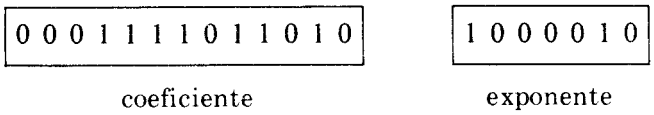
El punto decimal se interpreta en la representación de un número de la siguiente manera:

$$c \cdot r^e$$

donde c representa el contenido del registro coeficiente y e el contenido del registro exponente. El radical (base) r y la posición del punto radical en el coeficiente se asumen siempre. Considérese por ejemplo, un computador que asume representación de enteros para el coeficiente y base 8 para el exponente. El número octal $+17.32 = +1732 \times 8^{-2}$ se verá como sigue:



Cuando la representación octal se convierte a binaria, el valor binario del registro se convierte en:



Un número de punto flotante se dice que es *normalizado* si la posición más significativa del coeficiente contiene un dígito diferente de cero. De esta forma el coeficiente no tiene ceros por delante y contiene el máximo número posible de dígitos significativos. Considérese por ejemplo un registro coeficiente que puede acomodar cinco dígitos decimales y un signo. El número $+.00357 \times 10^3 = 3.57$ no es normalizado porque tiene dos ceros por adelante y el coeficiente no normalizado tiene una precisión hasta de tres dígitos significativos. El número puede normalizarse desplazando el coeficiente dos posiciones a la izquierda y disminuyendo el exponente en 2 para obtener: $+.35700 \times 10^1 = 3.5700$, el cual tiene una precisión hasta cinco dígitos significativos.

Las operaciones aritméticas con una representación de números de punto flotante son más complicadas que las operaciones aritméticas con números de punto fijo y su ejecución se demora más tiempo y requiere materiales más complejos. Sin embargo, la representación de punto flotante es más conveniente debido a los problemas de graduación envueltos en las operaciones de punto fijo. Muchos computadores tienen una capacidad interna para realizar operaciones aritméticas de punto flotante. Aquellos que no tienen esta facilidad se programan usualmente para operar de este modo.

Sumar o restar dos números en representación de punto flotante requiere primero una alineación del punto del radical, ya que la parte expo-

nencial debe hacerse igual antes de que los coeficientes se sumen o resten. Esta alineación se hace desplazando un coeficiente mientras que su exponente se ajusta hasta que sea igual al otro exponente. La multiplicación o división de punto flotante no requiere alineación del punto del radical. El producto puede formarse multiplicando los dos coeficientes y agregando los dos exponentes. La división se logra de la división con los coeficientes y la sustracción del exponente del divisor menos el exponente del dividendo.

8-10 DATOS NO NUMERICOS

Los tipos de datos considerados hasta ahora representan números que el computador usa como operandos para las operaciones aritméticas. Sin embargo, un computador no es una máquina que sólo almacena números y hace aritmética a alta velocidad. A menudo, un computador manipula símbolos en vez de números. La mayoría de programas escritos para los usuarios de computador están en forma de caracteres, es decir, un conjunto de símbolos que abarcan letras, dígitos y varios caracteres especiales. Un computador es capaz de aceptar caracteres (en código binario), almacenarlos en la memoria y realizar operaciones con los caracteres trasferidos a un componente de salida. Un computador puede funcionar como una máquina manipuladora de una cadena de caracteres. Por *cadena de caracteres* se implica una secuencia finita de caracteres escritos uno después de otro.

Los caracteres se representan en los registros del computador por medio de un código binario. En la Tabla 1-5, se listaron 3 códigos de caracter diferentes de uso común. Cada componente del código representa un carácter y consiste de seis, siete u ocho bits dependiendo del código. El número de caracteres que pueden ser almacenados en un registro depende de la longitud del registro y el número de bits usados en el código. Por ejemplo, un computador con una longitud de palabra de 36 bits que usa un código de 6 bits y puede almacenar seis caracteres por palabra. Las cadenas de caracteres se almacenan en la memoria en lugares consecutivos. El primer carácter en la cadena puede ser especificado a partir de la dirección de la primera palabra. El último carácter de la cadena puede encontrarse a partir de la dirección de la última palabra, o por especificación de una cuenta de caracteres, o por una marca especial que designa el final de la cadena de caracteres. La manipulación de caracteres se hace en los registros de la unidad de proceso con cada carácter representando una unidad de información.

Otros símbolos diferentes pueden ser almacenados en los registros del computador en forma de código binario. Un código binario puede ser adoptado para representar notas musicales para la producción de música por computador. Códigos binarios especiales son necesarios para representar patrones de lenguaje para un sistema automático de reconocimiento de lenguaje hablado. La representación de caracteres por medio de una matriz de puntos en pantalla CRT (tubo de rayos catódicos) requiere una representación en código binario por cada símbolo que se representa. La información de campo para supervisar la operación de un proceso contro-

lado o sistema de distribución de potencia usa información binaria codificada predeterminada. El tablero de ajedrez y las fichas para llevar a cabo un juego por computador requiere alguna forma de representación de la información en código binario.

Las operaciones hechas principalmente con datos numéricos son trasferencias, lógica, desplazamientos y decisiones de control. Las operaciones de trasferencia pueden preparar la información binaria codificada en algún orden requerido por la memoria y trasferir dicha información de y a las unidades externas. Las operaciones lógicas y de desplazamiento permiten una capacidad de realizar tareas de manipulación de datos para ayudar en el proceso de tomar decisiones.

Las microoperaciones lógicas son muy útiles para manipular bits individuales almacenados en un registro o un grupo de bits que conforman un símbolo binario-codificado dado. Las operaciones lógicas pueden cambiar valores de bits, eliminar un grupo de bits, o agregar nuevos valores de bits en un registro. Los siguientes ejemplos muestran cómo los bits de un registro se manipulan por lógica y microoperaciones de desplazamiento, como una función de operandos de lógica que están prealmacenados en la memoria.

La microoperación OR puede ser usada para poner a uno un bit o un grupo seleccionado de bits en un registro. Las relaciones de Boole $x + 1 = 1$ y $x + 0 = x$ determinan que la variable binaria x aplicada a una compuerta OR con un 1, produce un 1 independientemente del valor binario de x; pero, cuando se aplica a una compuerta OR con un 0, no cambiará el valor de x. Así, al aplicar a una compuerta OR un bit dado A_i de un registro con un 1, es posible poner a 1 el bit A_i sin tener en cuenta el valor previo. Considérese el siguiente ejemplo específico:

$$\begin{array}{ll} 0101\ 0101 & A \\ 1111\ 0000 & B \\ \hline 1111\ 0101 & A \leftarrow A \vee B \end{array}$$

El operando lógico en B tiene unos en las cuatro posiciones de los bits de mayor orden. Al aplicar a una compuerta OR este valor con el valor presente de A, es posible poner a 1 los cuatro bits de mayor orden de A pero dejando los cuatro bits de menor orden sin cambio. Así, la microoperación OR puede ser usada para establecer selectivamente los bits de un registro.

La operación AND puede ser usada para borrar un bit o un grupo seleccionado de bits de un registro. Las relaciones de Boole $x \cdot 0 = 0$ y $x \cdot 1 = x$ implican que la variable binaria x una vez aplicada con un 0 a una compuerta AND producirá un 0 independientemente del valor binario de x; pero, cuando se aplica con un 1 a una compuerta AND no cambiará el valor de x. Un bit A_i dado en el registro A puede ser llevado a 0 si se aplica con un 0 a una compuerta AND. Considérese un operando lógico $B = 0000\ 1111$. Cuando este operando se aplica conjuntamente con los contenidos de un registro a una compuerta AND, borrará los cuatro bits de mayor orden del registro pero dejarán los cuatro bits sin cambiar:

$$\begin{array}{ll} 0101\ 0101 & A \\ 0000\ 1111 & B \\ \hline 0000\ 0101 & A \leftarrow A \wedge B \end{array}$$

La microoperación AND puede usarse para borrar selectivamente los bits de un registro. La operación AND se llama algunas veces una operación de *máscara* porque enmascara o remueve todos los unos de una porción seleccionada de un registro.

La operación AND seguida de una operación OR puede usarse para cambiar un bit de un grupo de bits de un valor dado a un valor nuevo deseado. Esto se hace para enmascarar primero los bits y luego aplicar a una compuerta OR el nuevo valor. Por ejemplo, supóngase que un registro A contiene ocho bits, 0110 0101. Para remplazar los cuatro bits de mayor orden por el valor 1100, se enmascara primero los cuatro bits que no se requieren:

$$\begin{array}{ll} 0110\ 0101 & A \\ 0000\ 1111 & B1 \\ \hline 0000\ 0101 & A \leftarrow A \wedge B1 \end{array}$$

y luego se agrega el nuevo valor:

$$\begin{array}{ll} 0000\ 0101 & A \\ 1100\ 0000 & B2 \\ \hline 1100\ 0101 & A \leftarrow A \vee B1 \end{array}$$

La operación de máscara es una microoperación AND y la operación de inserción es una microoperación OR.

La microoperación XOR (OR-exclusiva) puede usarse para complementar un bit o un grupo seleccionado de bits de un registro. Las relaciones de Boole $x \oplus 1 = x'$ y $x \oplus 0 = x$ implican que la variable binaria x puede ser complementada cuando se aplica con un 1 a una compuerta OR-exclusiva y si es con un 0 permanece inalterable. Aplicando un solo bit de un registro con un 1 a una compuerta OR-exclusiva es posible complementar el bit seleccionado. Considérese el ejemplo numérico:

$$\begin{array}{ll} 1101\ 0101 & A \\ 1111\ 0000 & B \\ \hline 0010\ 0101 & A \leftarrow A \oplus B \end{array}$$

Los cuatro bits de mayor orden de A se complementan después de la operación OR-exclusiva con el operando B. La microoperación OR-exclusiva puede usarse para complementar selectivamente los bits de un registro. Si el operando B tiene solo unos, la operación OR-exclusiva complementará todos los bits de A. Si el contenido de A se aplica consigo mismo a una compuerta OR-exclusiva, se borrará el registro ya que $x \oplus x = 0$:

$$\begin{array}{ll} 0101\ 0101 & A \\ 0101\ 0101 & A \\ \hline 0000\ 0000 & A \leftarrow A \oplus A \end{array}$$

El valor de los bits individuales de un registro puede ser determinado enmascarando primero todos los bits excepto aquel en cuestión y luego comprobando si el registro es igual a 0. Supóngase que se requiere determinar si el bit 4 en el registro A es 0 ó 1:

$$\begin{array}{ll} 101x010 & A \\ 0001000 & B \\ \hline 000x000 & A \leftarrow B \wedge A \end{array}$$

El bit marcado x puede ser 0 ó 1. Cuando todos los demás bits estan enmascarados con el operando en B, el registro A contendrá sólo ceros si el bit 4 hubiera sido 0. Si el bit 4 originalmente fue un 1, este bit permanecerá en 1. Comprobando si el contenido de A es 0 ó no se determina si el bit cuatro fue 0 ó 1.

Si cada bit del registro debe comprobarse para 0 ó 1, es más conveniente desplazar el registro a la izquierda y trasferir el bit de mayor orden o un registro especial de 1 bit que comúnmente se llama el flip-flop del bit de arrastre. Después de cada desplazamiento, el arrastre puede comprobarse si es 0 ó 1 y se toma una decisión dependiendo del resultado.

Las operaciones de desplazamiento son útiles para agrupar o dispersar información binaria codificada. Agrupar información binaria tal como caracteres es una operación que une dos o más caracteres en una palabra. Dispersar es la operación inversa que separa dos o más caracteres almacenados en una palabra a caracteres individuales. Considérese la agrupación de dígitos BDC que se introdujeron primero como caracteres ASCII. El código de caracteres ASCII para los dígitos 5 y 9 se obtiene de la Tabla 1-5. Cada uno contiene siete bits y se coloca un 0 en la posición de mayor orden como se muestra a continuación. El carácter 5 se trasfiere al registro A, y el 9 al registro B. Los cuatro bits de mayor orden no tienen ningún uso para una representación BDC, de manera que se desenmascaran. El agrupamiento de dos dígitos BDC en el registro A consiste en desplazar el registro A cuatro veces a la izquierda (con ceros colocados en las posiciones de bits de menor orden) y luego aplicando a una compuerta OR el contenido de los registros:

	A	B
ASCII 5 =	0011 0101	0011 1001 = ASCII 9
AND con 0000 1111	0000 0101	0000 1001
Desplazar A a la izquierda cuatro veces	0101 0000	
$A \leftarrow A \vee B$	0101 1001 = BCD 59	

Una operación de desplazamiento con un 0 colocado en el bit del extremo se considera una microoperación de desplazamiento lógico.

La operación binaria disponible en un registro durante operaciones lógicas se llama una *palabra lógica*. Una palabra lógica se interpreta como una cadena de bits en oposición a una cadena de caracteres o datos numéricos. Cada bit en una palabra lógica funciona exactamente de la misma manera que otro bit cualquiera; en otras palabras, la unidad de información de una palabra lógica es un bit.

8-11 CODIGOS DE INSTRUCCION

La organización interna de un sistema digital se define por los registros que usa y la secuencia de microoperaciones que realiza con datos almacenados en los registros. En un sistema digital para *propósitos especiales*, la secuencia de microoperaciones se fija y el sistema ejecuta la misma tarea específica una y otra vez. Un computador digital es un sistema digital para *propósitos generales* capaz de ejecutar varias operaciones y además, puede recibir instrucciones sobre la secuencia específica de operaciones que debe realizar. El usuario de un computador puede controlar el proceso por medio de un *programa*, es decir, un conjunto de instrucciones que especifican las operaciones, operandos y la secuencia en la cual el procesamiento tiene que ocurrir. La tarea de procesamiento de datos puede ser alterada simplemente especificando un nuevo programa con diferentes instrucciones o especificando las mismas instrucciones con datos diferentes. Los códigos de instrucción, conjuntamente con los datos, se almacenan en la memoria. El control lee cada instrucción de la memoria y la localiza en el registro de control. El control interpreta entonces la instrucción y procede a ejecutarla emitiendo, una secuencia de funciones de control. Cada computador para propósito general tiene su propio repertorio único de instrucciones. La habilidad de almacenar y ejecutar instrucciones, el concepto de programa almacenado, es la propiedad más importante de un computador para propósito general.

Un *código de instrucción* es un grupo de bits que le dice al computador cómo realizar una operación específica. Por lo general se divide en dos partes, cada una conteniendo su propia interpretación particular. La parte más básica de un código de instrucción es su parte operativa. El *código de operación* de una instrucción es un grupo de bits que define una operación tal como sumar, restar, multiplicar, desplazar y complementar. El conjunto de operaciones de máquina formulados por un computador depende del procedimiento que se intenta llevar a cabo. El número total de operaciones así obtenidas determina el conjunto de operaciones de máquina. El número de bits requeridos para la parte de operación del código de instrucción es una función del número total de operaciones usadas. Debe consistir de por lo menos n bits para 2^n (o menos) operaciones dadas diferentes. El diseñador asigna una combinación de bits (un código) a cada operación. La unidad de control del computador se diseña para aceptar esta configuración de bits en el tiempo adecuado en una secuencia y suministrar las señales de comando adecuadas, a los destinos determinados, para poder ejecutar la operación específica. Como ejemplo específico, con-

sidérese un computador que usa 32 operaciones distintas, una de ellas siendo una operación de SUMA. El código de operación puede consistir de cinco bits con una configuración de bits 10010 asignada a la operación de SUMA. Cuando el código de operación 10010 es detectado por la unidad de control, se aplica una señal de comando a un circuito sumador para sumar dos números.

La parte de operación de un código de instrucción especifica la operación que se va a realizar. Esta operación debe ejecutarse con algunos datos usualmente almacenados en los registros del computador. Un código de instrucción, por tanto, debe especificar no solamente la operación sino también los registros donde los operandos se encuentran de la misma manera que el registro donde el resultado se almacena. Estos registros deben especificarse en un código de instrucción de dos maneras. Se dice que un registro se especifica *explícitamente* si el código de instrucción contiene bits especiales para su identificación. Por ejemplo, una instrucción puede contener no solamente una parte de operación sino también una dirección de memoria. Se dice que una dirección de memoria especifica explícitamente un registro de memoria. Por otra parte un registro se especifica *implícitamente* si éste se incluye como parte de la definición de la operación, es decir, si el registro está implícito en la parte operativa del código.

Formatos de códigos de instrucción

El formato de una instrucción usualmente se dibuja en un recuadro rectangular simbolizando los bits de la instrucción a medida que ellos aparecen en las palabras de la memoria o en un registro de control. Los bits de la instrucción se dividen algunas veces en grupos que subdividen la instrucción en partes. A cada grupo se le crea un nombre simbólico tal como parte del *código de operación* o parte de una *dirección*. Las diferentes partes especifican diferentes funciones para la instrucción y cuando se muestran juntas constituyen un formato de código instrucción.

Considérese por ejemplo, los tres formatos de código instrucción especificados en la Figura 8-12. El formato de instrucción en (a) consiste de un código de operación que implica un registro en la unidad procesadora. Se puede usar para especificar operaciones tales como "borrar el registro del procesador", o "completar un registro", o "trasferir el contenido de un registro a un segundo registro". El formato de instrucción en (b) tiene un código de operación seguido de un operando. Este se llama una instrucción de operando *inmediato,* porque el operando sigue inmediatamente después de la parte del código de operación de la instrucción. Se puede usar para especificar operaciones tales como "sumar el operando al contenido presente del registro" o "trasferir el operando al registro procesador", o puede especificar cualquier otra operación a ejecutar entre el contenido de un registro y un operando dado. El formato de instrucción especificado en la Figura 8-12(c) es similar al de (b) excepto que el operando debe extraerse de la memoria al lugar especificado por la parte de dirección de la instrucción. En otras palabras, la operación especificada por el código de operación se hace entre un registro procesador y un ope-

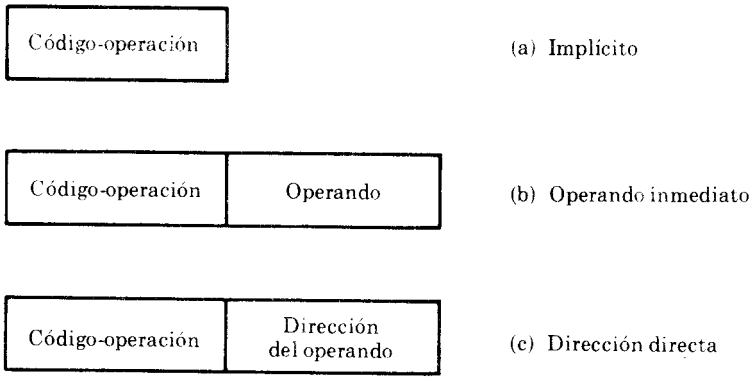


Figura 8-12 Tres formatos posibles de instrucción

rando que puede almacenarse en la memoria de alguna manera. La dirección de este operando en la memoria se incluye en la instrucción.

Asúmase que se tiene una unidad de memoria con 8 bits por palabra y que un código de operación contiene 8 bits. La localización de los tres códigos de la instrucción en la memoria se dibuja en la Figura 8-13. En la dirección 25 se tiene una instrucción implícita que especifica una operación: "trasferir el contenido del registro procesador R al registro procesador A". Esta operación puede ser simbolizada por la proposición:

$$A \leftarrow R$$

En las direcciones de memoria 35 y 36 se tiene una instrucción de operando inmediato que ocupa dos palabras. La primera palabra en la dirección 35 es el código de operación para la instrucción "trasfiera el operando al registro A", simbolizado como:

$$A \leftarrow \text{operando}$$

El operando mismo se almacena inmediatamente después del código de operación en la dirección 36.

En las direcciones 45 y 46, hay una instrucción de dirección directa que especifica la operación:

$$A \leftarrow M[\text{dirección}]$$

Esta simboliza una operación de trasferencia de memoria de un operando, el cual se especifica por la parte de dirección de la instrucción. La segunda palabra de la instrucción en la *dirección* 46 contiene la *dirección* y su valor es el binario 70. Por tanto, el operando a trasferirse al registro A, es el almacenado en la dirección 70 y su valor se muestra como el binario 28. Nótese que la instrucción se almacena en la memoria en alguna dirección. Esta instrucción tiene una parte de dirección que da la dirección del operando. Para evitar la confusión al decir la palabra "dirección" tantas

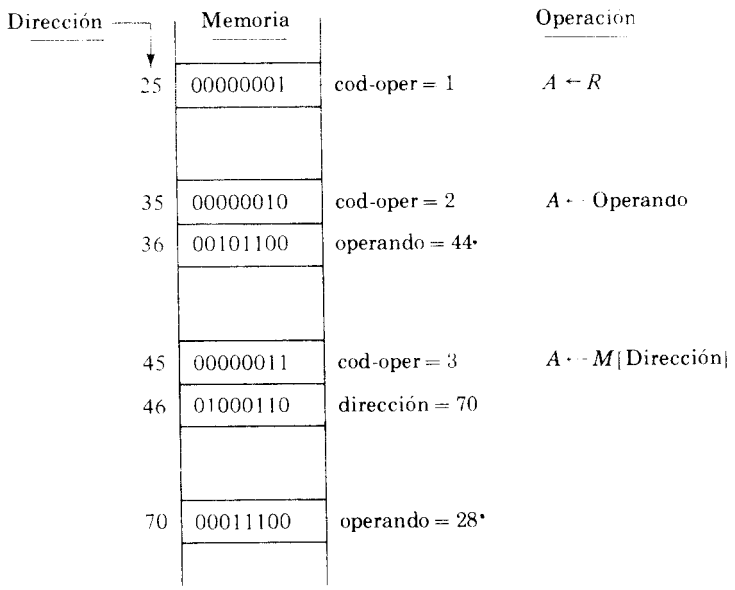


Figura 8-13 Representación de la memoria de instrucciones

veces, es costumbre referirse a la dirección de memoria como una "localización". Así la instrucción de dirección directa se almacena en las localizaciones 45 y 46. La dirección del operando en la 46 y el operando está disponible en la 70.

De debe tener en cuenta que la colocación de las instrucciones en la memoria como se muestra en la Figura 8-13 es una de las muchas alternativas. Solamente los computadores muy pequeños tienen palabras de 8 bits. Los computadores de gran tamaño pueden tener de 16 a 24 bits por palabra. En la mayoría de los computadores la instrucción completa puede agruparse en una palabra, y en algunos aun, se pueden agrupar dos o más instrucciones en una sola palabra de memoria.

Los formatos de instrucción mostrados en la Figura 8-12 son tres de los muchos formatos posibles que pueden formularse para computadores digitales. Se presentan aquí como un ejemplo y no deben considerarse como las únicas posibilidades. Los Capítulos 11 y 12 presentan y analizan otras instrucciones y formatos de códigos de instrucción.

En este punto, se debe reconocer la relación entre una *operación* y una *microoperación* de la manera aplicada a un computador digital. Una operación se especifica por una instrucción almacenada en la memoria de un computador. Es un código binario que dice al computador que realice una operación específica. La unidad de control recibe la instrucción de la memoria e interpreta los bits del código de operación. Esta envía entonces una secuencia de funciones de control para realizar microoperaciones en los registros internos del computador.

Por cada instrucción en la memoria, que especifica una operación, el control envía una secuencia de microoperaciones que se necesitan para la configuración de los componentes de un código de operación específico.

Una operación es especificada por el usuario en la forma de instrucción al computador. Una microoperación es una operación elemental que está restringida por los materiales disponibles dentro del computador.

Macrooperaciones versus microoperaciones

Hay ocasiones en que es conveniente expresar una secuencia de microoperaciones en una sola proposición. Una proposición que requiere una secuencia de microoperaciones para su configuración se llama una *macrooperación*. Una proposición en el método de notación de trasferencia entre registros, que define una instrucción, es una proposición de *macrooperación*, aunque las proposiciones de macrooperación de igual manera pueden usarse en otros casos. El método de trasferencia entre registros puede usarse para definir la operación especificada por una instrucción de computador, ya que todas las instrucciones especifican alguna operación de trasferencia entre registros, para que ésta última sea ejecutada por los componentes del computador.

Al observar una declaración de trasferencia entre registros aisladamente no se puede decir si ésta representa una macro o microoperación ya que ambos tipos de proposiciones denotan alguna proposición de trasferencia entre registros. La única manera de distinguir entre ellas es reconocer a partir del contenido y los componentes internos del sistema en cuestión, si la proposición se ejecuta con una función de control o no. Si la proposición puede ser ejecutada con una función de control sencilla, ésta representa una microoperación. Si la ejecución de la proposición por medio de los componentes, requiere dos o más funciones de control, se tomará la proposición como una microoperación. Solamente si se conocen las restricciones de los componentes del sistema se puede contestar esta pregunta.

Considérese, por ejemplo, la instrucción de la Figura 8-13 simbolizada por medio de la proposición:

$$A \leftarrow \text{operando}$$

Esta proposición es una macrooperación porque ésta especifica una instrucción de computador. Para ejecutar la instrucción la unidad de control debe emitir funciones de control para la siguiente secuencia de microoperaciones:

1. Leer el código de operación de la dirección 35.
2. Trasferir el código de operación al registro de control.
3. El control decodifica el código de operación y los reconoce como una instrucción de operando inmediato, de manera que lea la operación de la dirección 36.
4. El operando leído de la memoria se trasfiere al registro A.

La microoperación del paso 4 ejecuta la instrucción, pero los pasos 1 a 3 son necesarios antes de ella para que el control interprete la instrucción en sí.

La proposición que simboliza la instrucción:

$$A \leftarrow R$$

es también una macrooperación porque el control tiene primero que leer el código de operación en la dirección 25 para decodificarlo y reconocerlo. La trasferencia entre registros en sí se ejecuta con una segunda función de control.

El método de trasferencia entre registros es adecuado para describir las operaciones entre los registros en un sistema digital. Se puede usar en diferentes niveles de presentación si se tiene en cuenta que se interpreten las proposiciones adecuadamente. Se puede usar específicamente para las siguientes tareas.

1. Definir instrucciones de computador de una manera concisa por medio de proposiciones de macrooperación.
2. Expresar cualquier operación deseada por medio de una proposición de macrooperación sin ninguna relación con una configuración específica de componentes.
3. Definir la organización interna de los sistemas digitales por medio de funciones de control y microoperaciones.
4. Diseñar un sistema digital especificando los componentes de los materiales y sus interconexiones.

El conjunto de instrucciones para un computador dado puede explicarse en palabras, pero cuando se define con proposiciones de macrooperación, puede establecerse la definición precisamente con un mínimo de ambigüedad. El uso de otras proposiciones de macrooperación puede facilitar las especificaciones iniciales de un sistema y las proposiciones pueden usarse para simular el sistema cuando se desea comprobar la operación que se requiere. La organización interna de un sistema digital se describe de mejor manera por medio de un conjunto de funciones de control y microoperaciones. La lista de proposiciones de trasferencia entre registros que describe la organización del sistema, puede usarse para deducir las funciones digitales con las cuales se puede diseñar el sistema.

La siguiente sección muestra un ejemplo de cómo el método de trasferencia entre registros se usa en cada una de las cuatro tareas listadas anteriormente. Esto se hace al definir y diseñar un computador muy sencillo.

8-12 DISEÑO DE UN COMPUTADOR SENCILLO

El diagrama de bloque de un computador sencillo se muestra en la Figura 8-14. El sistema consiste de una unidad de memoria, siete registros y dos decodificaciones. La unidad de memoria tiene 256 palabras de 8 bits cada una, lo cual constituye poca capacidad para un computador real pero suficiente para demostrar las operaciones básicas encontradas en la mayoría de los computadores. Las instrucciones y los datos se almacenan en la unidad de memoria, pero todo el proceso de información se hace en los

registros. Los registros se listan en la Tabla 8-4, conjuntamente con una breve descripción de su función y el número de bits que contienen.

El registro de dirección de memoria *MAR*, almacena la dirección de la memoria. El registro separador de memoria *MBR* almacena el contenido de la palabra de memoria leída o escrita en la memoria. Los registros *A* y *R* son registros del procesador para propósito general.

El contador del programa *PC*, el registro de instrucción *IR* y el contador de tiempo *T*, son parte de la unidad de control. El *IR* recibe el código de operación de instrucciones. El decodificador asociado con este registro suministra una salida para cada código de operación encontrado. Así $q_1 = 1$, si el código de operación es el binario 1, $q_2 = 1$ si el código de operación es el binario 2 y así sucesivamente. El contador *T* se decodifica también para suministrar ocho variables de tiempo, t_0 hasta t_7 (ver Sección 7-6). Este contador se incrementa con cada pulso de reloj, pero puede borrarse en cualquier momento para comenzar una nueva secuencia desde t_0.

El *PC* pasa por una secuencia de cuenta paso a paso y causa que el computador dé las instrucciones sucesivas almacenadas previamente en la memoria. El *PC* siempre almacena la dirección de la siguiente instrucción en la memoria. Para leer una instrucción, el contenido de *PC* se trasfiere al *MAR* y se inicia un ciclo de lectura de memoria. El *PC* se incrementa en 1 de tal manera que almacene la siguiente dirección en la secuencia de instrucciones. Un código de operación leído de la memoria al *MBR*, se tras-

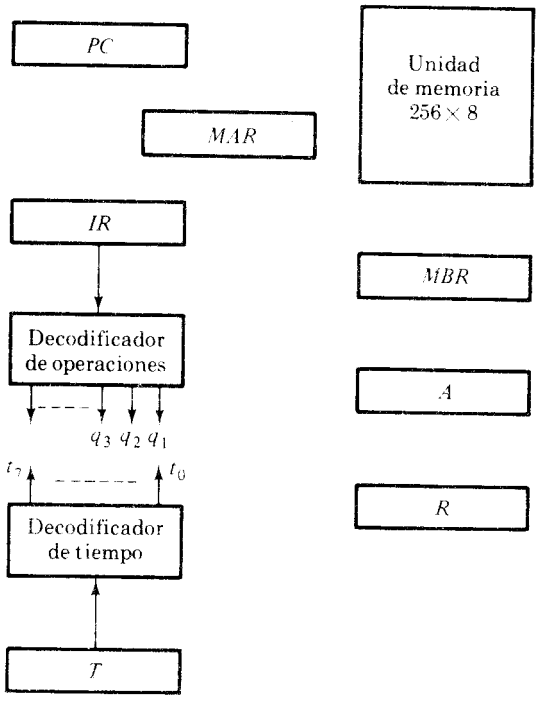


Figura 8-14 Diagrama de bloque de un computador simple

Tabla 8-4 Lista de registros para un computador sencillo

Símbolo	Número de bits	Nombre del registro	Función
MAR	8	Registro de dirección de memoria	Almacena direcciones de memoria
MBR	8	Registro separador de memoria	Almacena contenidos de palabras de memoria
A	8	Registro *A*	Registro procesador
R	8	Registro *R*	Registro procesador
PC	8	Contador de programa	Almacena la dirección de instrucción
IR	8	Registro de instrucción	Almacena códigos de operación corrientes
T	3	Contador de tiempo	Generador de secuencias

Tabla 8-5 Tres instrucciones para un computador sencillo

Código de operación	Mnemónico		Descripción	Función
00000001	MOV	R	Mover R a A	$A \leftarrow R$
00000010	LDI	OPRD	Cargar OPRD a A	$A \leftarrow$ OPRD
00000011	LDA	ADRS	Cargar el operando especificado por ADRS a A	$A \leftarrow M$[ADRS]

fiere al *IR*. Si la parte de dirección de memoria de una instrucción se lee al *MBR*, esta dirección se trasfiere al *MAR* para leer el operando. Así, el *MAR* puede recibir direcciones del PC o del *MBR*.

Las tres instrucciones definidas en la sección previa se especifican de nuevo en la Tabla 8-5. Como hay ocho bits en el código de operación, es posible especificar hasta 256 operaciones diferentes. Para simplificar la presentación se considera aquí solamente las tres instrucciones listadas. La mnemotecnia asociada con cada instrucción puede usarse por los programadores para especificar las instrucciones con nombres simbólicos. La sigla MOV (move) se establece para la operación del código binario correspondiente y simboliza una instrucción de "movimiento". El símbolo R por delante de MOV indica que el contenido de R se mueve al registro A. La sigla mnemónica LDI (load inmediate) simboliza una instrucción de carga inmediata. El OPRD en seguida de LDI se establece para un operando actual que el programador debe especificar con esta instrucción. LDA (load into A) es una abreviatura para "cargar a A" y ADRS a continuación establece para un número de dirección que el programador debe especificar con esta instrucción. Los valores actuales del OPRD y ADRS conjuntamente con su código de operación correspondiente se almacenarán en la memoria como en la Figura 8-13.

La Tabla 8-5 da una descripción en palabras para cada instrucción. Esta descripción en palabras no es muy precisa. Las proposiciones lista-

das bajo la columna de función dan una definición precisa y concisa de cada instrucción.

Un computador con solamente tres instrucciones no es muy útil. Se debe asumir que este computador tiene muchas más instrucciones aunque se consideren tres de ellas. Un programa escrito para el computador se almacena en la memoria. Este programa consiste de muchas instrucciones, pero de vez en cuando la instrucción usada será una de las tres listadas. Se consideran ahora las operaciones internas necesarias para ejecutar las instrucciones que están almacenadas en la memoria.

Ciclo de envío de instrucciones

El contador de programa PC debe inicializarse con lo contenido en la primera dirección del programa almacenado en la memoria. Cuando se activa el interruptor de "comienzo", la secuencia del computador sigue un patrón básico. Un código de operación cuya dirección está en el PC se lee de la memoria al MBR. El PC se incrementa en 1 para prepararla para la siguiente dirección en secuencia. El código de operación se trasfiere del MBR al IR donde es decodificado por el control. Esta secuencia se llama ciclo de *envío* de instrucción, ya que ésta saca el código de operación de la memoria y lo coloca en un registro de control. Las variables de tiempo, t_0, t_1 y t_2 que salen del decodificador de tiempos se usan como funciones de control para darle secuencia a las microoperaciones para leer un código de operación (op-code) y colocarlo en el IR:

t_0:	$MAR \leftarrow PC$	trasferir dirección del cod. de operación
t_1:	$MBR \leftarrow M$, $PC \leftarrow PC + 1$	leer el cod. de operación, incrementar PC
t_2:	$IR \leftarrow MBR$	transferir el cod. de operación al IR

Se asume que el contador de tiempo T comienza a partir del valor 000, el cual produce una variable de tiempo t_0 que sale del decodificador. El registro T se incrementa con cada pulso de reloj y automáticamente produce la siguiente variable de tiempo en la secuencia. Las tres primeras variables de tiempo ejecutan las secuencias de microoperación las cuales pueden simbolizarse por medio de la proposición de macrooperación:

$$IR \leftarrow M[PC], \quad PC \leftarrow PC + 1$$

Esta establece que la palabra de memoria especificada por la dirección en el PC se trasfiere al IR y luego se incrementa el PC. La restricción de los componentes en el computador sencillo es que solamente el MAR y el MBR pueden comunicarse con la memoria. Como el PC y el IR no pueden comunicarse directamente con la memoria, la anterior macrooperación debe ejecutarse con una secuencia de tres microoperaciones. Otra restricción de los materiales es que el PC no puede incrementarse mientras que su valor se use para suministrar la dirección para una lectura de memoria. Solamente después de que se complete una operación de lectura puede

incrementarse el PC. Al trasferir el contenido del PC al MAR, puede ser incrementado el PC mientras que la memoria lee la palabra direccionada por el MAR.

El ciclo de envío es común a todas las instrucciones. Las microoperaciones y funciones de control que preceden al ciclo de envío se determinan en la sección de control a partir del código de operación decodificado. Este está disponible de las salidas q_i, $i = 1, 2, 3, \ldots$ en el decodificador de operación.

Ejecución de las instrucciones

Durante la variable de tiempo t_3, el código de operación está en el IR y una salida del decodificador de operación es igual a 1. El control usa las variables q_i para determinar las siguientes microoperaciones en secuencia. La instrucción MOV R tiene un código de operación que hace $q_1 = 1$. La ejecución de esta instrucción requiere la microoperación:

$$q_1t_3: \quad A \leftarrow R, T \leftarrow 0$$

Así, cuando $q_1 = 1$ en el tiempo t_3, el contenido de R se trasfiere al registro A y el registro de tiempo T se borra. Borrado T, el control regresa a producir la variable de tiempo t_0 y así comenzar de nuevo el ciclo de envío, para leer el código de operación de la siguiente instrucción en secuencia. Recuérdese que PC se incrementa durante el tiempo t_1, de manera que mantiene la dirección de la siguiente instrucción en secuencia.

La instrucción LDI OPRD tiene un código de operación que hace $q_2 = 1$. Las microoperaciones que ejecutan esta instrucción son:

q_2t_3:	$MAR \leftarrow PC$	trasferir dirección del operando
q_2t_4:	$MBR \leftarrow M, PC \leftarrow PC + 1$	leer el operando, incrementar PC
q_2t_5:	$A \leftarrow MBR, T \leftarrow 0$	trasferir el operando, pasar al ciclo de envío.

Las tres variables de tiempo que siguen el ciclo de envío mientras que $q_2 = 1$ leen el operando de la memoria y lo trasfieren al registro A. Como el operando está en un lugar de la memoria en seguida del código de operación, se lee de la memoria a partir de la dirección especificada por el PC. El operando leído al MBR se trasfiere entonces a A. Nótese que el PC se incrementa una vez más para prepararlo para la dirección del siguiente código de operación antes de regresar al ciclo de envío.

La instrucción LDA ADRS tiene un código de operación que hace $q_3 = 1$. Las microoperaciones necesarias para ejecutar esta instrucción se listan a continuación:

q_3t_3:	$MAR \leftarrow PC$	trasferir la siguiente dirección de instrucción
q_3t_4:	$MBR \leftarrow M, PC \leftarrow PC + 1$	leer DIRECCION, (ADRS) incrementar PC
q_3t_5:	$MAR \leftarrow MBR$	trasferir dirección del operando

q_3t_6: $MBR \leftarrow M$ leer el operando

q_3t_7: $A \leftarrow MBR, T \leftarrow 0$ trasferir el operando a A, pasar al ciclo de envío

La dirección del operando, simbolizada por ADRS, se coloca en la memoria después del código de operación. Como el PC fue incrementado en t_1 durante el ciclo de envío éste mantiene la dirección donde se almacena el ADRS. El valor del ADRS se lee de la memoria en el tiempo t_4. Se incrementa PC durante este tiempo para prepararlo para el ciclo de envío de la siguiente instrucción. En el tiempo t_5, se trasfiere el valor del ADRS del MBR al MAR. Como el ADRS especifica la dirección del operando, una lectura de memoria durante el tiempo t_6 causará que el operando se establezca en el MBR. El operando del MBR se trasfiere al registro A y el control regresa al ciclo de envío.

Las funciones de control y microoperaciones para un computador sencillo se resumen en la Tabla 8-6. Las primeras tres variables de tiempo constituyen el ciclo de envío mediante las cuales se lee el código de operación hacia el IR. Las microoperaciones que se ejecutan durante el tiempo t_3 dependen del valor del código de operación en el IR. Hay tres funciones de control que son funciones de t_3, pero q_1 ó q_2 ó q_3 puede ser igual a 1 durante t_3. La microoperación particular ejecutada durante el tiempo t_3, es aquella cuya función de control correspondiente tiene una variable q que es igual a 1. Lo mismo puede decirse de las otras variables de tiempo.

Un computador práctico tiene muchas instrucciones y cada instrucción requiere un ciclo de envío para leer un código de operación. Las microoperaciones necesarias para la ejecución de las instrucciones particulares se especifican mediante las variables de tiempo y por la q_i particular, $i = 0, 1, 2, 3, \ldots, 255$, que sucede estar en el estado 1 durante este tiempo. La lista de funciones de control y las microoperaciones para un computador práctico deberían ser mayores que las mostradas en la Tabla 8-6. Obviamente, el simple computador no es un elemento práctico, pero usando solamente tres instrucciones se pueden demostrar claramente las funcio-

Tabla 8-6 Proposiciones de trasferencia entre registros para un computador sencillo

ENVIAR	t_0:	$MAR \leftarrow PC$
	t_1:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	t_2:	$IR \leftarrow MBR$
MOVER	q_1t_3:	$A \leftarrow R, T \leftarrow 0$
CARGA INMED.	q_2t_3:	$MAR \leftarrow PC$
	q_2t_4:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	q_2t_5:	$A \leftarrow MBR, T \leftarrow 0$
CARGA A A	q_3t_3:	$MAR \leftarrow PC$
	q_3t_4:	$MBR \leftarrow M, PC \leftarrow PC + 1$
	q_3t_5:	$MAR \leftarrow MBR$
	q_3t_6:	$MBR \leftarrow M$
	q_3t_7:	$A \leftarrow MBR, T \leftarrow 0$

nes básicas de un computador digital. La extensión de este principio al computador con más instrucciones y más registros de procesador debería ser aparente a partir de este ejemplo. En el Capítulo 11 se usan los principios presentados aquí para diseñar un computador más real.

Diseño del computador

Se había mostrado anteriormente que la lógica de trasferencia entre registros es adecuada para definir las operaciones especificadas por las instrucciones del computador. Se ha demostrado justamente que la lógica de trasferencia entre registros es un método conveniente para especificar la secuencia de funciones internas en un computador digital, conjuntamente con las microoperaciones que ellas ejecutan. Se mostrará ahora que la lista de funciones de control y microoperaciones para un sistema digital es un punto de comienzo conveniente para el diseño del sistema. La lista de microoperaciones especifica el tipo de registros y las funciones digitales asociadas que deben ser incorporadas en el sistema. La lista de funciones de control especifican las compuertas lógicas requeridas para la unidad de control. Para demostrar este procedimiento se estudiará el diseño del computador sencillo a partir de la lista de proposiciones de trasferencia entre los registros dados en la Tabla 8-6.

El primer paso en el diseño es repasar las proposiciones de trasferencia entre registros, listadas en la Tabla 8-6 y escoger todas aquellas proposiciones que realizan la misma macrooperación en el mismo registro. Por ejemplo, la microoperación $MAR \leftarrow PC$ se lista en la primera línea con la función de control t_0, en la quinta línea con la función de control $q_2 t_3$ y en la octava línea con la función de control $q_3 t_3$. Las tres líneas se combinan en una sola proposición:

$$t_0 + q_2 t_3 + q_3 t_3: \quad MAR \leftarrow PC$$

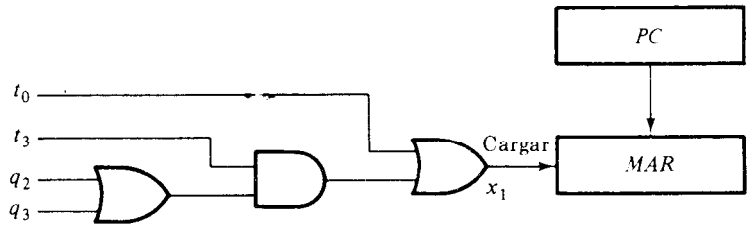


Figura 8-15 Configuración de x_1: $MAR \leftarrow PC$

Recuérdese que una función de control es una función de Boole. El + entre las funciones de control denotan una operación de Boole OR, y la secuencia de un operador entre q_2 y t_3 denota una operación de Boole AND. La anterior proposición combina todas las condiciones de control para la trasferencia de PC hasta MAR. La configuración de los componentes de la proposición anterior se dibuja en la Figura 8-15. La función de control puede ser manipulada como una función de Boole para dar:

$$x_1 = t_0 + q_2t_3 + q_3t_3 = t_0 + (q_2 + q_3)t_3$$

La variable binaria x_1 se aplica a la entrada de carga del *MAR* y las salidas del *PC* se aplican a las entradas del *MAR*. Cuando $x_1 = 1$, el siguiente pulso de reloj trasfiere el contenido del *PC* al *MAR*. Las variables binarias que causan que x_1 sea 1 vienen de los decodificadores de operación de tiempo de la unidad de control.

Hay ocho microoperaciones diferentes listadas en la Tabla 8-6. Para cada microoperación distinta, se acumularán las funciones de control asociadas y se aplicarán conjuntamente a una compuerta OR. El resultado es como se muestra en la Tabla 8-7. Las funciones de control obtenidas para cada microoperación se forman en una ecuación de la variable binaria x_i, $i = 1, 2, \ldots, 8$. Las ocho variables x pueden ser generadas fácilmente con las compuertas AND y OR pero no se harán aquí.

El diseño de un computador sencillo se puede obtener de la información de trasferencia entre registros dada en la Tabla 8-7. El diagrama de bloque diseñado se muestra en la Figura 8-16. Aquí se tienen de nuevo los siete registros, la unidad de memoria y los dos decodificadores. Además, hay un recuadro marcado "circuito combinacional". El bloque del circuito combinacional genera las ocho funciones de control, x_1 hasta x_8, de acuerdo a la lista de funciones de control de la tabla. Las funciones de control habilitan la carga e incrementan las entradas de varios registros. Un registro que recibe información de dos fuentes necesita un multiplexor para seleccionar entre los dos. Por ejemplo, el *MAR* recibe información del *MBR* o del *PC*. El multiplexor asociado con el *MAR* trasfiere el contenido del *PC* cuando su línea seleccionada es un 1 ($x_1 = 1$) pero trasfiere el contenido del *MBR* cuando la línea seleccionada es 0. Esto es debido a que $x_1 = 0$, cuando $x_2 = 1$, pero x_2 inicia la entrada de carga del *MAR*, de manera que el contenido del *MBR* pasa por el multiplexor hasta el *MAR*. El contador de tiempo T se incrementa con cada pulso de reloj; sin embargo, cuando $x_7 = 1$ se borrará y colocará a 0.

Los registros y otras funciones digitales especificadas en la Figura 8-16 pueden ser designadas individualmente por medio de procedimientos combinacionales y de lógica secuencial. Si el sistema se construye con circuitos integrados, se pueden encontrar circuitos MSI para todos los registros y funciones digitales. El circuito combinacional para el control

Tabla 8-7 Especificación de los componentes para un computador sencillo

$x_1 = t_0 + q_2t_3 + q_3t_3$:	$MAR \leftarrow PC$
$x_2 = q_3t_5$:	$MAR \leftarrow MBR$
$x_3 = t_1 + q_2t_4 + q_3t_4$:	$PC \leftarrow PC + 1$
$x_4 = x_3 + q_3t_6$:	$MBR \leftarrow M$
$x_5 = q_2t_5 + q_3t_7$:	$A \leftarrow MBR$
$x_6 = q_1t_3$:	$A \leftarrow R$
$x_7 = x_5 + x_6$:	$T \leftarrow 0$
$x_8 = t_2$:	$IR \leftarrow MBR$

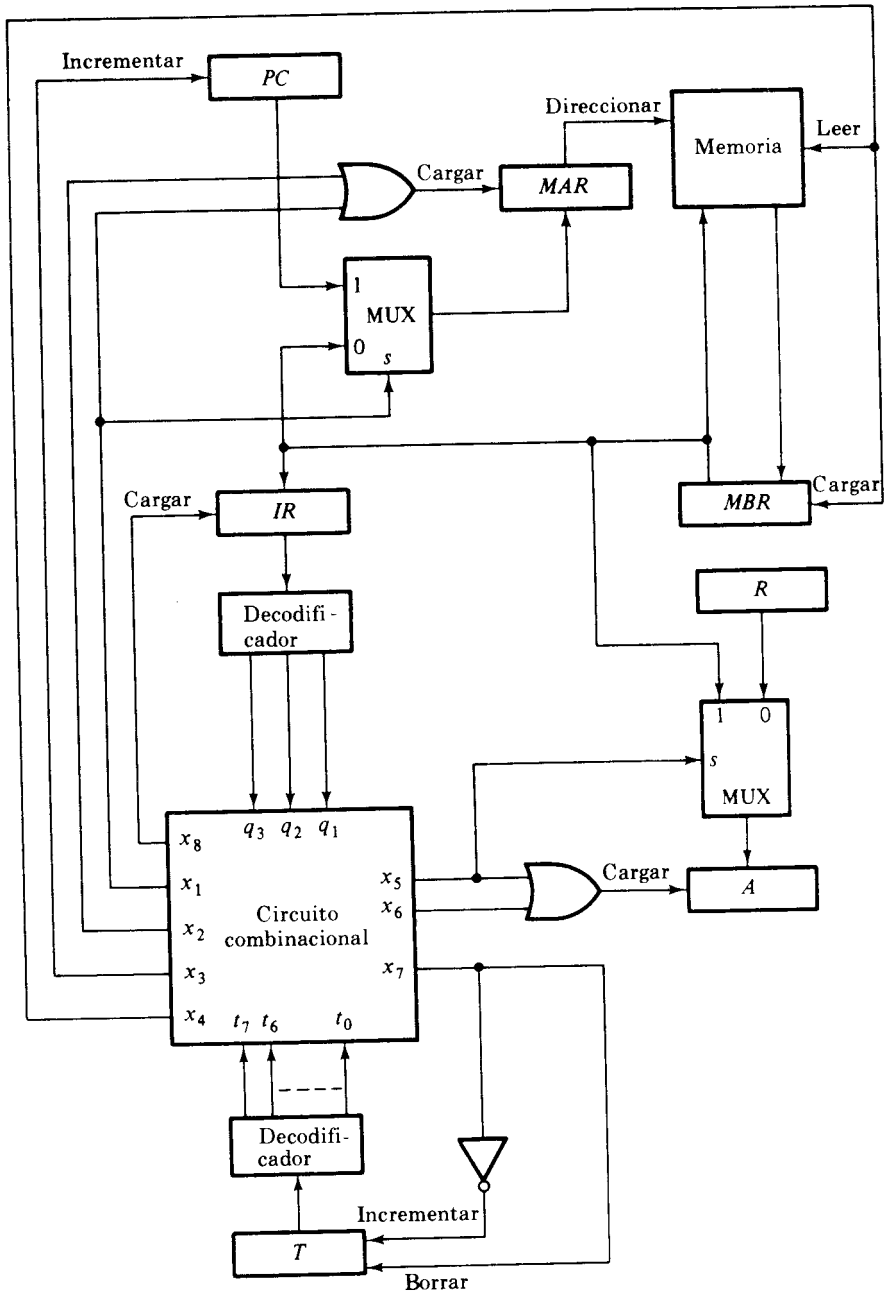


Figura 8-16 Diseño de un computador sencillo