Contenido

Prefacio

La lógica digital trata de la interconexión entre componentes digitales y módulos y en un término usado para denotar el diseño y análisis de los sistemas digitales. El ejemplo más conocido de un sistema digital es un computador digital para propósito general. Este libro presenta los conceptos básicos usados en el diseño y análisis de los sistemas digitales e introduce los principios de la organización del computador digital y su diseño. Presenta varios métodos y técnicas adecuados para una variedad de aplicaciones de diseño del sistema digital. Cubre todos los aspectos del sistema digital desde los circuitos de compuertas electrónicas hasta la estructura compleja de un sistema de microcomputador.

Los Capítulos 1 hasta 6 presentan técnicas de diseño de lógica de diseño desde el punto de vista *clásico*. El álgebra de Boole y las tablas de verdad se usan para el análisis y diseño de los circuitos combinacionales y las técnicas de transición de estado para el análisis y diseño de los circuitos secuenciales. Los Capítulos 7 hasta el 12 presentan métodos de diseño de sistemas digitales desde el punto de vista de *trasferencia entre registros*. El sistema digital se descompone en subunidades de registros y el sistema se especifica con una lista de proposiciones de trasferencia entre registros que describen las trasferencias operacionales de la información almacenada en los registros. El método de trasferencia entre registros se usa para el análisis y diseño de las unidades del procesador, unidades de control, un procesador central de computador y para describir las operaciones internas de microprocesadores y microcomputadores. El Capítulo 13 trata de la electrónica de los circuitos digitales y presenta las familias lógicas digitales más comunes a base de circuitos integrados.

Los componentes usados para construir sistemas digitales se fabrican en la forma de circuitos integrados. Los circuitos integrados contienen una gran cantidad de circuitos digitales interconectados dentro de una pequeña pastilla. Los dispositivos (MSI) de integración a mediana escala conforman funciones digitales y los dispositivos (LSI) de integración a gran escala conforman módulos de computador completos. Es muy importante para el diseñador lógico, familiarizarse con los diferentes componen-

tes digitales encontrados en la forma de circuitos integrados. Por esta razón muchos circuitos MSI y LSI se introducen a lo largo del libro y se explican completamente sus familias lógicas. El uso de circuitos integrados en el diseño de sistemas digitales se ilustra por medio de ejemplos en el texto y en los problemas al final de los capítulos.

Este libro fue planeado originalmente como una segunda edición del *diseño lógico de computadores,* del autor (Prentice-Hall, 1972). Debido a la gran cantidad de material nuevo y a las revisiones extensas que se han llevado a cabo, parece más apropiado adoptar un nuevo título para el texto presente. Alrededor de un tercio del texto es material que aparece en el libro anterior. Las otras dos terceras partes constituyen información nueva o revisada. Los factores fundamentales para las revisiones y adiciones surgen de las desarrolladas en la tecnología electrónica digital. Se da un gran énfasis a los circuitos MSI y LSI y a los métodos de diseño que usan circuitos integrados. El libro cubre varios componentes LSI de la variedad de grupo de bits y microcomputador. Presenta aplicaciones de la memoria de sólo lectura (ROM) y del arreglo lógico programable (PLA). Sin embargo, los adelantos posteriores en el método de diseño de trasferencia entre registros, demanda una nueva redacción de la segunda parte del libro.

El Capítulo 1 presenta varios sistemas binarios adecuados para representar información en componentes digitales. El sistema de números binarios se explica y se ilustran los códigos binarios para demostrar la representación de la información decimal y alfanumérica. La lógica binaria se introduce desde un punto de vista intuitivo antes de proceder con una definición formal del álgebra de Boole.

Los postulados básicos y teoremas del álgebra de Boole se encuentran en el Capítulo 2. Se enfatiza la correlación entre las expresiones de Boole y sus compuertas de interconección equivalentes. Todas las operaciones lógicas posibles para dos variables se investigan y a partir de ello se deducen las compuertas digitales disponibles en la forma de circuitos integrados se presentan al comienzo de este capítulo, pero se deja para la última parte del capítulo el análisis más detallado para describir la construcción interna de las compuertas.

El Capítulo 3 presenta el mapa y los métodos de tabulado para simplificar las funciones de Boole. El método del mapa se usa para simplificar circuitos digitales construidos con AND, OR, NAND, NOR, y compuertas lógicas alambradas. Los diferentes procesos de simplificación se sumarizan en forma de tabla para una referencia fácil.

Los procedimientos de diseño y análisis de los circuitos combinacionales se presentan en el Capítulo 4. Algunos componentes básicos usados en el diseño de sistemas digitales, tales como sumadores y convertidores de código son introducidos como ejemplos de análisis y diseño. El capítulo investiga configuraciones posibles usando circuitos combinacionales de multinivel NAND y NOR.

El Capítulo 5 versa sobre los componentes MSI y LSI de lógica combinacional. A menudo se explican funciones tales como sumadores paralelos, decodificadores y multiplexores, y se ilustra con ejemplos su uso en el diseño de circuitos combinacionales. La memoria de sólo lectura (ROM) y el arreglo lógico programable (PLA) son introducidos y se demuestra su utilidad en el diseño de circuitos combinacionales complejos.

El Capítulo 6 esboza varios métodos para el diseño y análisis de los circuitos secuenciales temporizados. El capítulo comienza presentando varios tipos de flip-flops y la forma como ellos son disparados. El diagrama de estado, tabla de estado, y las ecuaciones de estado se presentan como herramientas convenientes para analizar los circuitos secuenciales. Los métodos de diseño presentados, trasforman el circuito secuencial a un grupo de funciones de Boole que especifican la entrada lógica a los flip-flops del circuito. Las funciones de entrada de Boole se derivan de la tabla de excitación y se simplifican por medio de mapas.

En el Capítulo 7, se presentan una variedad de registros, registros de desplazamiento y contadores similares a aquéllos disponibles en la forma de circuitos integrados. Se explica la operación de la memoria de acceso aleatorio (RAM). Las funciones digitales introducidas en este capítulo son los bloques de construcción básicos a partir de los cuales se pueden construir sistemas digitales más complejos.

El Capítulo 8 introduce un método de trasferencia entre registros para describir los sistemas digitales. Este muestra cómo expresar en forma simbólica la secuencia de operación entre los registros de un sistema digital. Se definen símbolos para trasferencia entre registros, microoperaciones aritméticas, lógicas y de desplazamiento. Se cubren en detalle los diferentes tipos de datos almacenados en los registros de los computadores. Se usan algunos ejemplos típicos para mostrar cómo se presentan las instrucciones de computador en forma binaria codificada y cómo las operaciones especificadas por instrucciones pueden ser expresadas con proposiciones de trasferencia entre registros. El capítulo concluye con el diseño de un computador muy sencillo para demostrar el método de trasferencia entre registros del diseño de sistemas digitales.

El Capítulo 9 tiene que ver con la unidad procesadora de los computadores digitales. Se discuten alternativas para organizar una unidad procesadora con buses y memorias tapón (Scratchpad memory). Se presenta una unidad lógica, aritmética típica (ALU) y se desarrolla para el diseño de cualquier otra configuración de ALU. Se presentan también otros componentes encontrados comúnmente en los procesadores, tales como registros de condición y desplazamiento. Se comienza el diseño de un registro acumulador para propósitos generales, comenzando a partir de un grupo de operaciones de trasferencia entre registros y culminando con un diagrama lógico.

En el Capítulo 10 se introducen cuatro métodos de diseño de lógica de control. Dos de los métodos constituyen un control alambrado con circuito impreso. Los otros dos introducen el concepto de la microprogramación y cómo diseñar un controlador con un arreglo lógico programable (PLA). Los cuatro métodos son demostrados por medio de ejemplos que muestran el desarrollo de algoritmos de diseño y el procedimiento para obtener los circuitos de control del sistema. La última sección introduce un secuenciador de microprograma LSI y muestra cómo se puede usar en el diseño de una unidad de control de microprograma.

El Capítulo 11 está dedicado al diseño de un computador digital pequeño. Los registros en el computador son definidos y se especifica el conjunto de instrucciones del computador. La descripción del computador se

formaliza con las proposiciones de trasferencia entre registros que especifican las microoperaciones entre los registros, lo mismo que las funciones de control que inician esas microoperaciones. Se muestra entonces que el conjunto de microoperaciones puede usarse para diseñar la parte procesadora de datos del computador. Las funciones de control en la lista de proposiciones de trasferencia entre registros, suministran la información para el diseño de la unidad de control. La unidad de control para el computador se diseña por medio de tres métodos diferentes: el control alambrado con circuito impreso, el control PLA y el control del microprograma.

El Capítulo 12 es enfocado sobre varios componentes LSI para formar un sistema de microcomputador. La organización de un microprocesador típico se describe y explica su organización interna. Un conjunto típico de instrucciones para el microprocesador se presenta y se explican varios modos de direccionamiento. La operación de una pila y el manipuleo de las subrutinas e interrupciones, se cubre desde el punto de vista de los materiales. El capítulo ilustra también la conexión de las pastillas de memoria al sistema de bus del microprocesador y la operación de varias unidades de interconexión que se comunican con dispositivos de entrada y salida. Concluye con una descripción del modo de trasferencia de acceso directo a la memoria.

El Capítulo 13 detalla los circuitos electrónicos de la compuerta básica en siete familias lógicas de circuitos integrados. Este capítulo final debe ser considerado como un apéndice, puede ser omitido si se desea. El Capítulo 13 asume un conocimiento previo de electrónica básica, pero no hay un prerrequisito específico para el resto del libro.

Cada capítulo incluye un grupo de problemas y una lista de referencias. Las respuestas a los problemas seleccionados aparecen en el apéndice para suministrar una ayuda al estudiante y para ayudar al lector independiente. Un *manual de soluciones* se suministra para el instructor por parte del publicista.

El libro es adecuado para un curso en lógica digital y diseño de computadores en un departamento de ingeniería eléctrica o de computadores. Se puede usar también en un departamento de ciencia de computadores para un curso en organización de computador. Las partes del libro pueden usarse de varias formas: (1) Como un primer curso en lógica digital o circuitos de conmutación al cubrir los Capítulos 1 hasta el 7 y posiblemente el Capítulo 13. (2) Como un segundo curso, en lógica de computador digital con un prerrequisito de un curso en circuitos de conmutación básicos, basado en los Capítulos 3 y 7 hasta el 12. (3) Como una introducción a la configuración con materiales de los microprocesadores y microcomputadores al cubrir los Capítulos 8 hasta el 12.

En conclusión, me gustaría explicar la filosofía fundamental del material presentado en este libro. El método clásico ha sido predominante en el pasado para describir las operaciones de los circuitos digitales. Con el advenimiento de los circuitos integrados y especialmente de la introducción de los componentes LSI del microcomputador, el método clásico parece estar bastante lejos de las aplicaciones prácticas. Aunque el método clásico para describir sistemas digitales complejos no es directamente aplicable, el concepto básico de álgebra de Boole, lógica combinacional y procedimien-

to de lógica secuencial, son todavía importantes para comprender la construcción interna de muchas funciones digitales. Por otra parte, el método de trasferencia entre registros, presenta una mejor representación para describir las operaciones entre los diferentes módulos en los sistemas digitales. Este versa de la trasferencia de cadenas de bits en paralelo y puede ser considerado como de un nivel mayor en la jerarquía de la representación del sistema digital. La transición del método clásico al de trasferencia entre registros, se hace en este libro por medio de las funciones MSI de circuitos integrados. Los Capítulos 5 y 7 cubren muchas funciones digitales que están disponibles en circuitos integrados. Su operación se explica en términos de conpuertas y flip-flops que conforman el circuito digital particular. Cada circuito MSI se considera como una unidad funcional que realiza una función particular. Esta operación se describe en el método de rotación de trasferencia entre registros. Así, el análisis y diseño de registros y otras funciones digitales se hace por medio del método clásico, pero el uso de esas funciones al describir las operaciones de un sistema digital, se especifica por medio de proposiciones de trasferencia entre registros. El método de trasferencia entre registros se usa para definir las instrucciones de computador, para expresar las operaciones digitales en forma concisa, para demostrar la organización de los computadores digitales y para especificar los componentes de los materiales para el diseño de sistemas digitales.

Deseo expresar mis agradecimientos al Dr. John L. Fike por revisar el manuscrito original y al Profesor Víctor Payse por indicar correcciones durante la enseñanza del curso al usar el manuscrito. La mayor parte del trabajo de mecanografía fue hecho por Mrs. Lucy Albert y su hábil ayuda es apreciada grandemente. Mis mayores agradecimientos los doy a mi señora por las sugerencias que ella hizo al mejorar la facilidad de lectura del libro y por su ánimo y apoyo durante la preparación de éste.

M. MORRIS MANO

Sistemas binarios

1

1-1 COMPUTADORES DIGITALES Y SISTEMAS DIGITALES

Los computadores digitales han hecho posible muchos avances científicos, industriales y comerciales que no se hubiesen podido lograr por otros medios. Nuestro programa espacial hubiese sido imposible sin la vigilancia continua de tiempo real del computador y muchas empresas de negocios funcionan eficientemente sólo con la ayuda del procesamiento automático de datos. Los computadores se usan para cálculos científicos, procesamientos de datos comerciales y de negocios, control de tráfico aéreo, dirección espacial, campo educacional y en muchas otras áreas. La propiedad más impactante de un computador es su generalidad. Puede seguir una serie de instrucciones, llamadas *programa,* que operan con datos dados. El usuario puede determinar y cambiar los programas y datos de acuerdo a una necesidad específica. Como resultado de esta flexibilidad, los computadores digitales de uso general pueden realizar una serie de tareas de procesamiento de información de amplia variedad.

El computador digital de uso general es el ejemplo más conocido de sistema digital. Otros ejemplos incluyen conmutadores telefónicos, voltímetros digitales, contadores de frecuencia, máquinas calculadoras y máquinas teletipos. Típico de un sistema digital es su manejo de *elementos discretos* de información. Tales elementos discretos pueden ser impulsos eléctricos, los dígitos decimales, las letras de un alfabeto, las operaciones aritméticas, los símbolos de puntuación o cualquier otro conjunto de símbolos significativos. La yuxtaposición de elementos discretos de información representan una cantidad de información. Por ejemplo, las letras *d, o* y *g* forman la palabra *dog.* Los dígitos 237 forman un número. De la misma manera una secuencia de elementos discretos forman un lenguaje, es decir una disciplina que con lleva información. Los primeros computadores fueron usados principalmente para cálculos numéricos, en este caso los elementos discretos usados son los dígitos. De esta aplicación ha surgido el término *computador digital.* Un nombre más adecuado para un computador digital podría ser "sistema de procesamiento de información discreta".

Los elementos discretos de información se representan en un sistema digital por cantidades físicas llamadas *señales*. Las señales eléctricas tales como voltajes y corrientes son las más comunes. Las señales en los sistemas digitales electrónicos de la actualidad tienen solamente dos valores discretos y se les llama *binarios*. El diseñador de sistemas digitales está restringido al uso de señales binarias debido a la baja confiabilidad de los circuitos electrónicos de muchos valores. En otras palabras puede ser diseñado un circuito con diez estados que use un valor de voltaje discreto para cada estado, pero que tenga poca confiabilidad de operación. En contraste, un circuito de transistor que puede estar en conducción o corte tiene dos valores de señales posibles y puede ser construido para ser extremadamente confiable. Debido a la restricción física de los componentes y a que la lógica humana tiende a ser binaria, los sistemas digitales que estén restringidos a usar valores discretos, lo estarán para usar valores binarios.

Las cantidades discretas de información podrían desprenderse de la naturaleza del proceso o podrían ser cuantificadas a propósito de un proceso continuo. Por ejemplo, un programa de pago es un proceso discreto inherente que contiene nombres de empleados, números de seguro social, salarios semanales, impuestos de renta, etc. El cheque de pago de un empleado, se procesa usando valores discretos, tales como las letras de un alfabeto (nombres), dígitos (salarios) y símbolos especiales tales como $. Por otra parte, un científico investigador podría observar un proceso continuo pero anotar solamente cantidades específicas en forma tabular. El científico estará cuantificando sus datos continuos. Cada número en su tabla constituye un elemento discreto de información.

Muchos sistemas físicos pueden ser descritos matemáticamente por medio de ecuaciones diferenciales cuyas soluciones, como funciones de tiempo, darán un comportamiento matemático del proceso. Un *computador análogo* realiza una *simulación* directa de un sistema físico. Cada sección del computador es el análogo de alguna parte específica del proceso sometido a estudio. Las variables en el computador análogo están representadas por señales continuas que varían con el tiempo y que por lo general son voltajes eléctricos. Las señales variables son consideradas análogas con aquellas del proceso y se comportan de la misma manera. De esta forma, las mediciones de voltajes análogos pueden ser sustituidos por variables del proceso. El término *señal análoga* se sustituye por *señal continua* debido a que un "computador análogo" se ha convertido significativamente en un computador que maneja variables continuas.

Para simular un proceso físico en un computador digital, deben ser cuantificadas las cantidades. Una vez que las variables del proceso sean representadas por señales continuas de tiempo real, estas últimas serán cuantificadas por un aparato de conversión de análogo a digital. Un sistema físico, cuyo compartamiento se exprese por medio de ecuaciones matemáticas, se simula en un computador digital con base en métodos numéricos. Cuando el problema que va a ser procesado es inherentemente discreto, como en el caso de aplicaciones comerciales, el computador digital manipula las variables en su forma natural.

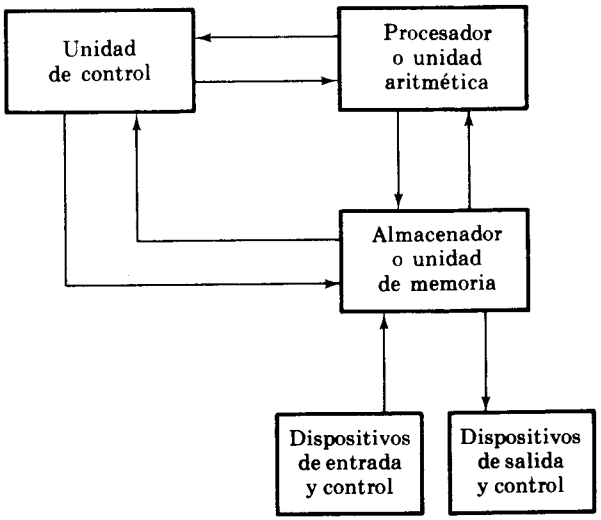


Figura 1-1 Diagrama de bloque de un computador digital

Un diagrama de bloque del computador digital se muestra en la Figura 1-1. La unidad de memoria almacena los programas de la misma forma que los datos de entrada, salida e intermedios. La unidad de proceso realiza tareas aritméticas y de procesamiento de datos según sea especificado por el programa. La unidad de control supervisa el flujo de información entre las diferentes unidades. Dicha unidad recupera las instrucciones una a una del programa acumulado en la memoria. Para cada instrucción, ella informa al procesador a fin de ejecutar la operación específica de la instrucción. Tanto el programa como los datos se almacenan en la memoria. La unidad de control supervisa el programa de instrucciones, y el procesador manipula los datos de acuerdo a las especificaciones del programa.

El programa y los datos preparados por el usuario son trasferidos a la unidad de la memoria mediante un elemento de entrada tal como una lectora de tarjetas perforada o una teleimpresora. Un elemento de salida tal como un impresor recibe el resultado de los cálculos y le presenta al usuario los resultados impresos. Los elementos de entrada y salida son sistemas digitales especiales manejables por partes electromecánicas y controladas por circuitos electrónicos digitales.

Una calculadora electrónica es un sistema digital similar al computador digital que tiene como elemento de entrada el teclado y como elemento de salida una pantalla numérica. Las instrucciones son trasferibles a la calculadora por medio de las teclas de función tales como el más y el menos. Los datos se introducen mediante las teclas numéricas y los resultados se muestran por pantalla en forma de números. Algunas calculadoras tienen algo de parecido a las computadoras digitales ya que tienen forma de imprimir y además facilidad de programación.

Un computador digital es sin embargo un aparato más poderoso que una calculadora; puede usar muchos otros dispositivos de entrada y salida, puede realizar no solamente cálculos aritméticos y operaciones lógicas sino que puede ser programado para tomar decisiones basadas en condiciones internas y externas.

Un computador digital es una interconexión de módulos digitales. Para poder comprender la operación de cada módulo digital es necesario tener los conocimientos básicos de los sistemas digitales y de su comportamiento. La primera mitad de este libro versa sobre sistemas digitales en general proporcionando los conocimientos necesarios para su diseño. La segunda mitad del libro trata sobre los diferentes módulos de un computador digital, su operación y su diseño. Las características operacionales de la unidad de memoria se explican en el Capítulo 7. La organización y diseño de la unidad de proceso se tratan en el Capítulo 9. Varios métodos para diseñar la unidad de control se introducen en el Capítulo 10. La organización y diseño de un computador digital completo y pequeño se presenta en el Capítulo 11.

Un procesador combinado con la unidad de control forma un componente llamado *unidad central de proceso* o CPU. Un CPU encapsulado en una pastilla de circuito integrado se denomina *microprocesador*. La unidad de memoria, de la misma forma que la parte que controla la interconexión entre el microprocesador y los elementos de entrada y salida, puede ser encapsulada dentro de la pastilla del microprocesador o puede encontrarse en pastillas pequeñas de circuitos integrados. Un CPU combinado con una memoria y un control de interconexión formará un computador de tamaño pequeño denominado *microcomputador*. La disponibilidad de los componentes del microcomputador ha revolucionado la tecnología de diseño de los sistemas digitales, permitiendo al diseñador la libertad de crear estructuras que antes eran antieconómicas. Los diferentes componentes de un sistema de microcomputador se representan en el Capítulo 12.

Ya se ha mencionado el hecho de que un computador digital manipula elementos discretos de información y que estos elementos se presentan en forma binaria. Los operandos, usados en los cálculos pueden ser expresados en el sistema de números binarios. Otros elementos discretos, incluidos los dígitos decimales, se representan con códigos binarios. El procesamiento de datos se lleva a cabo por medio de los elementos lógicos binarios, usando señales binarias. Las cantidades se acumulan en los elementos de almacenamiento binario. El propósito de este capítulo es el de introducir los diferentes conceptos binarios como marco de referencia para un posterior estudio de los capítulos restantes.

1-2 NUMEROS BINARIOS

Un número decimal tal como 7392 representa una cantidad igual a 7 unidades de mil, más 3 centenas, más 9 decenas, más 2 unidades. Las unidades de mil, las centenas, etc., son potencias de 10 implícitamente indicadas por la posición de los coeficientes. Para ser más exactos, 7392 puede ser escrito así:

$$7 \times 10^3 + 3 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

Sin embargo, lo convencional es escribir solamente los coeficientes y a partir de su posición deducir las potencias necesarias de 10. En general, un número con punto decimal puede ser representado por una serie de coeficientes de la siguiente manera:

$$a_5a_4a_3a_2a_1a_0\,,\,a_{-1}a_{-2}a_{-3}$$

Los coeficientes a_j son uno de los diez dígitos (0, 1, 2, . . ., 9) y el suscrito j da el lugar y por tanto el valor de la potencia de 10 por el cual debe ser multiplicado el coeficiente.

$$10^5a_5 + 10^4a_4 + 10^3a_3 + 10^2a_2 + 10^1a_1 + 10^0a_0 + 10^{-1}a_{-1} + 10^{-2}a_{-2} + 10^{-3}a_{-3}$$

Se dice que el sistema de números decimales tiene la *base* o *raíz* 10 debido a que usa diez dígitos y que los coeficientes son multiplicados por potencias de 10. El sistema *binario* es un sistema numérico diferente. Los coeficientes del sistema de números binarios tienen dos valores posibles: 0 y 1. Cada coeficiente a_j se multiplica por 2^j. Por ejemplo, el equivalente decimal del número binario 11010,11 es 26,75 como se demuestra de la multiplicación de los coeficientes por potencias de 2.

$$1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} = 26{,}75$$

En general, un número expresado en un sistema de base r tiene coeficientes multiplicados por potencias de r:

$$a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \cdots + a_2 \cdot r^2 + a_1 \cdot r + a_0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + \cdots + a_{-m} \cdot r^{-m}$$

Los coeficientes a_j varían en valor entre 0 y $r-1$. Para distinguir los números de bases diferentes, se encierran los coeficientes entre paréntesis y se escribe un suscrito igual a la base usada (con excepción en algunos casos de los números decimales en los cuales su contenido hace obvio que se trate de un decimal). Un ejemplo de un número de base 5 será:

$$(4021{,}2)_5 = 4 \times 5^3 + 0 \times 5^2 + 2 \times 5^1 + 1 \times 5^0 + 2 \times 5^{-1} = (511{,}4)_{10}$$

Nótese que los valores para coeficientes de base 5 pueden solamente ser 0, 1, 2, 3 y 4.

Es costumbre presentar los r dígitos necesarios para los coeficientes del sistema decimal en caso de que la base del número sea menor que 10. Las letras del alfabeto se usan para completar los diez dígitos decimales cuando la base del número sea mayor que 10. Por ejemplo, en el sistema de números *hexadecimal* (base 16) se presentan los primeros diez dígitos del sistema decimal. Las letras A, B, C, D, E y F se usan para los dígitos 10,

11, 12, 13, 14 y 15 respectivamente. Un ejemplo de números hexadecimal será:

$$(B65F)_{16} = 11 \times 16^3 + 6 \times 16^2 + 5 \times 16 + 15 = (46687)_{10}$$

Los primeros 16 números en los sistemas decimal, binario, octal y hexadecimal se listan en la Tabla 1-1.

Las operaciones aritméticas con números en base r siguen las mismas reglas que los números decimales. Cuando se usa una base diferente a la conocida de 10 se debe ser precabido de usar solamente las r dígitos permitidos. A continuación se muestran ejemplos de suma, resta y multiplicación de los números binarios:

sumando:	101101	minuendo:	101101	multiplicando:	1011
sumando:	+100111	sustraendo:	−100111	multiplicador:	×101
suma:	1010100	diferencia:	000110		1011
					0000
					1011
				producto:	110111

Tabla 1-1 Números con diferentes bases

Decimal (base 10)	Binario (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

La suma de dos números binarios se calcula mediante las mismas reglas que en decimales con la diferencia de que los dígitos de la suma en cualquier posición significativa pueden ser 0 ó 1. Cualquier "lleva" obtenida en una posición significativa dada, se usa por el par de dígitos en la posición significativa superior. La resta es un poco más complicada,

sus reglas son las mismas que en el caso del sistema decimal excepto que la "lleva" en una posición significativa dada agrega 2 al dígito del minuendo. (Una lleva en el sistema decimal agrega 10 al dígito del minuendo). La multiplicación es muy simple. Los dígitos del multiplicador son siempre 1 ó 0. Por tanto, los productos parciales son iguales al multiplicando o a 0.

1-3 CONVERSIONES ENTRE NUMEROS DE BASE DIFERENTE

Un número binario puede ser convertido a decimal formando la suma de las potencias de base 2 de aquellos coeficientes cuyo valor sea 1. Por ejemplo:

$$(1010{,}011)_2 = 2^3 + 2^1 + 2^{-2} + 2^{-3} = (10{,}375)_{10}$$

El número binario tiene cuatro unos y el decimal equivalente se deduce de la suma de cuatro potencias de 2. Similarmente, un número expresado en base r puede ser convertido a su equivalente decimal multiplicando cada coeficiente con su correspondiente potencia de r y sumando. El siguiente es un ejemplo de conversión de un sistema octal a decimal:

$$(630{,}4)_8 = 6 \times 8^2 + 3 \times 8 + 4 \times 8^{-1} = (408{,}5)_{10}$$

La conversión de decimal a binario o cualquier otro sistema de base r es más conveniente si el número se separa en *parte entera* y *parte fraccionaria* para hacer la conversión de cada parte separadamente. La conversión de un *entero* de sistema decimal o binario se explica de mejor manera en el siguiente ejemplo:

EJEMPLO 1-1: Convertir el decimal 41 a binario. Primero, 41 se divide por 2 para dar un cociente entero de 20 y un residuo de $\frac{1}{2}$. El cociente se divide a su turno por 2 para producir un cociente nuevo con su residuo. Se continua así el proceso hasta que el cociente entero se convierte en cero. *Los coeficientes* de los números binarios deseados se obtienen de los *residuos* de la siguiente manera:

	Cociente entero		*residuo*	*coeficiente*
$\frac{41}{2}$	$= 20$	$+$	$\frac{1}{2}$	$a_0 = 1$
$\frac{20}{2}$	$= 10$	$+$	0	$a_1 = 0$
$\frac{10}{2}$	$= 5$	$+$	0	$a_2 = 0$
$\frac{5}{2}$	$= 2$	$+$	$\frac{1}{2}$	$a_3 = 1$

cociente entero		*residuo*	*coeficiente*
$\frac{2}{2} = 1$	+	0	$a_4 = 0$
$\frac{1}{2} = 0$	+	$\frac{1}{2}$	$a_5 = 1$

respuesta: $(41)_{10} = (a_5 a_4 a_3 a_2 a_1 a_0)_2 = (101001)_2$

El proceso aritmético puede llevarse a cabo en forma más conveniente, de la siguiente manera:

entero	*residuo*	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = respuesta

La conversión de enteros decimales a cualquier sistema de base r es similar al ejemplo anterior con la diferencia de que la división se hace por r en vez de 2.

EJEMPLO 1-2: Convertir el decimal 153 a octal. La base requerida es 8. Primero se divide 153 por 8 para dar un cociente entero de 19 y un residuo de 1. Luego se divide 19 por 8 para dar un cociente entero de 2 y un residuo de 3. Finalmente, se divide 2 por 8 para dar un cociente de 0 y un residuo de 2. Este proceso puede hacerse convenientemente de la siguiente manera:

153	
19	1
2	3
0	2 $= (231)_8$

La conversión de una *fracción* decimal o binaria se lleva a cabo por un método similar al usado para enteros. Empero, se usa la multiplicación en vez de la división y se acumulan los enteros en vez de los residuos. El método se explica más claramente a continuación:

EJEMPLO 1-3: Convertir $(0{,}6875)_{10}$ a binario. Primero se multiplica 0,6875 por 2 para dar un entero y una fracción. La nueva fracción se multiplica por 2 para dar un número entero y una nueva fracción. Este proceso se continúa hasta que la fracción se convierta en 0 o hasta que el número de dígitos tenga la suficiente precisión. Los coeficientes del número binario se obtienen de los enteros de la siguiente manera:

	entero		*fracción*	*coeficiente*
$0{,}6875 \times 2 =$	1	+	0,3750	$a_{-1} = 1$
$0{,}3750 \times 2 =$	0	+	0,7500	$a_{-2} = 0$
$0{,}7500 \times 2 =$	1	+	0,5000	$a_{-3} = 1$
$0{,}5000 \times 2 =$	1	+	0,0000	$a_{-4} = 1$

respuesta: $(0{,}6875)_{10} = (0{,}a_{-1}a_{-2}a_{-3}a_{-4})_2 = (0{,}1011)_2$

Para convertir una fracción decimal a un número expresado en base r, se usa un procedimiento similar: se multiplica por r en vez de 2 y los coeficientes encontrados de los enteros varían entre valores desde 0 hasta $r-1$ en vez de 0 y 1.

EJEMPLO 1-4: Convertir $(0{,}513)_{10}$ a octal.

$$0{,}513 \times 8 = 4{,}104$$
$$0{,}104 \times 8 = 0{,}832$$
$$0{,}832 \times 8 = 6{,}656$$
$$0{,}656 \times 8 = 5{,}248$$
$$0{,}248 \times 8 = 1{,}984$$
$$0{,}984 \times 8 = 7{,}872$$

La respuesta con siete cifras significativas se obtiene de la parte entera de los productos:

$$(0{,}513)_{10} = (0{,}406517\ldots)_8$$

La conversión de números decimales con parte fraccionaria y entera, se hace convirtiendo la parte fraccionaria y la entera separadamente y luego combinando las dos respuestas. Usando los resultados de los Ejemplos 1-1 y 1-3 se obtiene:

$$(41{,}6875)_{10} = (101001{,}1011)_2$$

De los Ejemplos 1-2 y 1-4, se obtiene:

$$(153{,}513)_{10} = (231{,}406517)_8$$

1-4 NUMEROS HEXADECIMALES Y OCTALES

La conversión de binario a octal y hexadecimal y viceversa juega un papel muy importante en los computadores digitales. Como $2^3 = 8$ y $2^4 = 16$, cada dígito octal corresponde a tres dígitos binarios y cada dígito hexadecimal corresponde a cuatro dígitos binarios. La conversión de binario a octal se lleva a cabo fácilmente haciendo la partición del número binario en grupos de tres dígitos, cada uno comenzando desde el punto binario y haciéndolo de izquierda a derecha. El dígito octal correspondiente se asigna a cada grupo. El siguiente ejemplo es una ilustración del procedimiento:

$$(\underbrace{10}_{2}\ \underbrace{110}_{6}\ \underbrace{001}_{1}\ \underbrace{101}_{5}\ \underbrace{011}_{3} \cdot \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{000}_{0}\ \underbrace{110}_{6})_2 = (26153{,}7406)_8$$

La conversión de binario a hexadecimal es similar excepto que el número binario se divide en grupos de cuatro dígitos:

$$(\underbrace{10}_{2},\underbrace{1100}_{C}\ \underbrace{0110}_{6}\ \underbrace{1011}_{B} \cdot \underbrace{1111}_{F}\ \underbrace{0010}_{2})_2 = (2C6B,F2)_{16}$$

El dígito hexadecimal correspondiente para cada grupo de dígitos binarios es fácilmente recordado después de estudiar los valores listados en la Tabla 1-1.

La conversión de octal o hexadecimal a binario se hace por un procedimiento inverso al anterior. Cada dígito octal se convierte a un equivalente binario de tres dígitos. De la misma manera, cada dígito hexadecimal se convierte a un equivalente binario de cuatro dígitos. Esto se ilustra con ejemplos a continuación:

$$(673{,}124)_8 = (\underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{011}_{3} \cdot \underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{100}_{4})_2$$

$$(306, D)_{16} = (\underbrace{0011}_{3}\ \underbrace{0000}_{0}\ \underbrace{0110}_{6} \cdot \underbrace{1101}_{D})_2$$

Los números binarios son difíciles de trabajar ya que necesitan tres o cuatro veces más dígitos que su equivalente decimal. Por ejemplo, el número binario 111111111111 es equivalente al decimal 4095. Empero, los computadores digitales usan los números binarios y algunas veces se hace necesario que el operador humano o usuario se comunique directamente con la máquina en términos de números binarios. Un esquema que retiene el sistema binario en el computador pero que reduce el número de dígitos que el humano debe considerar, utiliza la relación que hay entre el sistema de números binarios y el sistema hexadecimal u octal. Mediante este método, el humano piensa en términos de números octales o hexadecimales y hace la conversión por medio de la inspección, cuando se hace necesaria la comunicación directa con la máquina. Así el número binario 111111111111 tiene 12 dígitos y se expresa en octal como 7777 (cuatro dígitos) o en hexadecimal como FFF (tres dígitos). Durante la comunicación de la gente (relativa a números binarios en el computador), se hace más deseable la representación hexadecimal u octal ya que puede ser usada de manera más compacta con una tercera o cuarta parte del número de dígitos necesarios para expresar el número binario equivalente. Cuando un humano se comunica con la máquina (a través de los interruptores de la consola, las luces indicadoras o por medio de los programas escritos en *lenguaje de máquina*), la conversión de octal o hexadecimal a binario y viceversa se hace por inspección de parte del usuario.

1-5 COMPLEMENTOS

Los complementos se usan en los computadores digitales para simplificar la operación de sustracción y para manipulaciones lógicas. Hay dos clases de complementos para cada sistema de base r: (1) El complemento de r y (2) el complemento de $(r-1)$. Cuando se sustituye el valor de la base los dos tipos reciben los nombres de complementos de 2 y 1 en el uso de los números binarios o complementos de 10 y 9 en el caso de los números decimales.

El complemento de r

Dado un número positivo N en base r con parte entera de n dígitos, se define el complemento r de N como $r^n - N$ para $N \neq 0$ y 0 para $N = 0$. El siguiente ejemplo numérico ayudará a comprender mejor la situación:

El complemento de 10 de $(52520)_{10}$ es $10^5 - 52520 = 47480$.

El número de dígitos del número es $n = 5$.

El complemento de 10 de $(0{,}3267)_{10}$ es $1 - 0{,}3267 = 0{,}6733$.

No hay parte entera, por tanto $10^n = 10^0 = 1$.

El complemento de 10 de $(25{,}639)_{10}$ es $10^2 - 25{,}639 = 74{,}361$.

El complemento de 2 de $(101100)_2$ es $(2^6)_{10} - (101100)_2$
$= (1000000 - 101100)_2 = 010100$.

El complemento de 2 de $(0{,}0110)_2$ es $(1 - 0{,}0110)_2 = 0{,}1010$.

Por la definición y los ejemplos, es claro que el complemento de 10 de un número decimal puede ser formado dejando todos los ceros menos significativos inalterados, restando el primer número diferente de cero menos significativo de 10 para luego sustraer el resto de dígitos más significativos de 9. El complemento de 2 puede ser formado dejando todos los ceros menos significativos y el primer dígito diferente de cero sin cambio, para luego remplazar unos por ceros y ceros por unos en el resto de dígitos mas significativos.

Un tercer método más sencillo para obtener el complemento de r es dado después de la definición del complemento de $(r-1)$. El complemento de r de un número existe para cualquier base r (siendo r mayor pero no igual a 1) y puede ser obtenido de la definición que se dará a continuación. Los ejemplos listados aquí usan números con $r = 10$ (decimal) y $r = 2$ (binario) debido a que estos son las bases más interesantes. El nombre del complemento se relaciona con la base del número usado. Por ejemplo el complemento de $(r-1)$ de un número en base 11 se llama complemento de 10 ya que $r - 1 = 10$ para $r = 11$.

El complemento de $(r - 1)$

Dado un número positivo N en base r con una parte entera de n dígitos y una parte fraccionaria de m dígitos, se define el complemento de $(r-1)$ de N como $r^n - r^{-m} - N$. Se dan algunos ejemplos a continuación:

El complemento de 9 de $(52520)_{10}$ es $(10^5 - 1 - 52520) = 99999 - 52520 = 47479$.

Como no hay parte fraccionaria, entonces $10^{-m} = 10^0 = 1$.

El complemento de 9 de $(0{,}3267)_{10}$ es $(1 - 10^{-4} - 0{,}3267) = 0{,}9999 - 0{,}3267 = 0{,}6732$.

Como no hay parte entera entonces $10^n = 10^0 = 1$.

El complemento de 9 de $(25{,}639)_{10}$ es $(10^2 - 10^{-3} - 25{,}639) = 99{,}999 - 25{,}639 = 74{,}360$.

El complemento de 1 de $(101100)_2$ es $(2^6 - 1) - (101100) = (111111 - 101100)_2 = 010011$.

El complemento de 1 de $(0{,}0110)_2$ es $(1 - 2^{-4})_{10} - (1{,}0110)_2 = (0{,}1111 - 0{,}0110)_2 = 0{,}1001$.

De estos ejemplos se ve que el complemento de 9 de un número decimal se forma simplemente sustrayendo cada dígito de 9. El complemento de 1 de un número binario se expresa en una forma aún más sencilla: los unos se cambian a ceros y los ceros a unos. Como el complemento de $(r-1)$ se puede obtener muy fácilmente el complemento de r. De las definiciones y de la comparación de los resultados obtenidos en los ejemplos se desprende que el complemento de r puede ser obtenido del complemento de $(r-1)$ después de sumar r^{-m} al dígito menos significativo. Por ejemplo el complemento de 2 de 10110100 se obtiene del complemento de 1 de 01001011 agregando 1 para dar 01001100.

Vale la pena mencionar que el complemento del complemento deja al número en su valor original. El complemento de r de N es $r^n - N$ y el complemento de $(r^n - N)$ es $r^n - (r^n - N) = N$; de la misma manera sucede con el complemento de 1.

Sustracción con complementos de r

El método directo de sustracción diseñado en las escuelas usa el concepto de prestar. En este método se presta un 1 de una posición significativa más alta cuando el dígito del minuendo es más pequeño que el correspondiente dígito del sustraendo. Esto parece el método más sencillo usado por la gente al hacer la sustracción con papel y lápiz. Cuando la sustracción se ejecuta por medio de los componentes digitales se encuentra que este método es menos eficiente que el método que usa complementos y suma de la forma descrita a continuación.

La sustracción de dos números positivos $(M-N)$, ambos en base r puede hacerse de la siguiente manera:

1. Se suma el minuendo M al complemento de r del sustraendo N.
2. Se inspeccionan los datos obtenidos en el Paso 1 para una "lleva" final.
   (a) Si ocurre una "lleva" final, se debe descartar.

(b) Si no ocurre una "lleva" final, se toma el complemento de r del número obtenido en el paso 1 y se coloca un número negativo al frente.

Los siguientes ejemplos ilustran el procedimiento:

EJEMPLO 1-5: Usando el complemento de 10, sustraer $72532 - 3250$.

$M = 72532$ 72532
$N = 03250$

complemento de 10 de $N = 96750$ + 96750

lleva final → 1 / 69282

respuesta: 69282

EJEMPLO 1-6: Sustraer: $(3250 - 72532)_{10}$.

$M = 03250$ 03250
$N = 72532$

complemento de 10 de $N = 27468$ + 27468

ninguna lleva / 30718

respuesta: $-69282 = -$ (complemento de 10 de 30718)

EJEMPLO 1-7: Usar el complemento de 2 para sustraer $M - N$ con los números binarios dados.

(a) $M = 1010100$ 1010100
$N = 1000100$

complemento de 2 de $N = 0111100$ + 0111100

lleva final → 1 / 0010000

respuesta: 10000

(b) $M = 1000100$ 1000100
$N = 1010100$

complemento de 2 de $N = 0101100$ + 0101100

ninguna lleva / 1110000

respuesta: $-10000 = -$ (complemento de 2 de 1110000)

La prueba de este procedimiento es: la suma de M al complemento de r de N da $(M+r^n-N)$. Para números que tienen una parte entera de N dígitos, r^n es igual a 1. (Lo que se ha llamado la "lleva" final) en la posición $(N+1)$. Como se asume que M y N son positivos, por tanto:

$$(a) \quad (M + r^n - N) \geqslant r^n \quad \text{si } M \geqslant N, \quad \text{o}$$
$$(b) \quad (M + r^n - N) < r^n \quad \text{si } M < N$$

En el caso (a) la respuesta es positiva e igual a $M-N$, y se obtiene directamente descartando la "lleva" final r^n. En el caso (b) la respuesta es negativa e igual a $-(N-M)$. Este caso se detecta por la ausencia de la "lleva" final. La respuesta se obtiene sacando un segundo complemento y agregando un signo negativo:

$$-[r^n - (M + r^n - N)] = -(N - M).$$

Sustracción con complemento de $(r - 1)$

El procedimiento para sustraer con el complemento de $(r-1)$ es exactamente el mismo que el usado con el complemento de r excepto por una variación llamada la "lleva" final de reinicio mostrada a continuación. La sustracción $M-N$ de dos números positivos en base r pueden calcularse de la siguiente manera:

1. Se agrega el minuendo M al complemento de $(r-1)$ del sustraendo N.
2. Se inspecciona el resultado en el Paso 1 y la "lleva" final.
   (a) Si aparece una "lleva" final se agrega 1 al dígito menos significativo (lleva final de reinicio).
   (b) Si no ocurre una "lleva" final, se obtiene el complemento de $(r-1)$ del número obtenido en el Paso 1 y se coloca un signo negativo al frente.

La prueba de este procedimiento es muy similar a la del complemento de r dada y se deja al lector como ejercicio. Los siguientes ejemplos ilustran este procedimiento:

EJEMPLO 1-8: Repetir los Ejemplos 1-5 y 1-6 usando complementos de

(a) $M = 72532$, $N = 03250$

complemento de 9 de $N = 96749$

$$\begin{array}{rr} & 72532 \\ + & 96749 \\ \hline & 1\,69281 \\ & \quad\quad + \\ & \quad\quad\;\; 1 \\ \hline & 69282 \end{array}$$

lleva final de reinicio

respuesta: 69282

(b) $M = 03250$ 03250
$N = 72532$
complemento de 9 de $N = 27467$ + 27467
ninguna lleva 30717

respuesta: $-69282 = -$ (complemento de 9 de 30717)

EJEMPLO 1-9: Repetir el Ejemplo 1-7 usando el complemento de 1.

(a) $M = 1010100$ 1010100
$N = 1000100$
complemento de 1 de $N = 0111011$ + 0111011
lleva final de reinicio 1 0001111
+
1
0010000

respuesta: 10000

(b) $M = 1000100$ 1000100
$N = 1010100$ +
complemento de 1 de $N = 0101011$ 0101011
ninguna lleva 1101111

respuesta: $-10000 = -$ (complemento de 1 de 1101111)

Comparación entre los complementos de 2 y de 1

Al comparar los complementos de 2 y de 1 se detallan las ventajas y desventajas de cada uno. El complemento de 1 es más fácil de ejecutar, por medio de componentes digitales ya que lo único que hay que hacer es cambiar los ceros a unos y los unos a ceros. La ejecución del complemento de 2 puede obtenerse de dos maneras: (1) agregando 1 al dígito significativo menor del complemento de 1 y (2) dejando los primeros ceros, en las posiciones significativas menores y el primer 1 inalterados para cambiar solamente el resto de unos a ceros y de ceros a unos. Durante la sustracción de los números, usando complementos, es ventajoso emplear el complemento de 2 en el cual solamente se requiere una operación aritmética de suma. El complemento de 1 requiere dos sumas aritméticas cuando sucede una "lleva" final de reinicio. El complemento de 1 tiene la desventaja adicional de poseer dos ceros aritméticos: uno con todos los ceros y otro con todos los

unos. Para ilustrar este hecho, considérese la sustracción de dos números binarios iguales $1100 - 1100 = 0$.

Usando el complemento de 1:

$$\begin{array}{rr} & 1100 \\ \pm & \\ & 0011 \\ \hline + & 1111 \end{array}$$

Complementar de nuevo para obtener -0000.

Usando el complemento de 2:

$$\begin{array}{rr} & 1100 \\ + & \\ & 0100 \\ \hline + & 0000 \end{array}$$

Mientras que el complemento de 2 tiene solamente un cero aritmético, el 0 complemento de 1 puede ser negativo o positivo lo cual podría complicar la situación.

Los complementos útiles para los cálculos aritméticos en los computadores se tratan en los Capítulos 8 y 9. El complemento de 1, sin embargo, es muy útil en los manipuladores lógicos (como se mostrará más adelante) ya que el cambio de unos a ceros y viceversa es equivalente a la operación de inversión lógica. El complemento de 2 se usa solamente en asocio de las aplicaciones aritméticas. En consecuencia es conveniente adoptar la siguiente convención: cuando se use la palabra *complemento,* sin mencionar el tipo, en asocio con una aplicación aritmética, se asume que es el complemento de 1.

1-6 CODIGOS BINARIOS

Los sistemas digitales electrónicos usan señales que tienen dos valores distintos y elementos de circuito que tienen dos estados estables. Existe una analogía directa entre las señales binarias, los elementos de circuito binarios y los dígitos binarios. Un número binario de r dígitos, por ejemplo, puede ser representado por n elementos de circuito binario con señales de salida equivalentes a 0 ó 1 respectivamente. Los sistemas digitales representan y manipulan no solamente los números binarios sino también muchos otros elementos directos de información. Cualquier elemento discreto de información específico entre un grupo de cantidades puede ser representado por un código binario. Por ejemplo el *rojo* es un color específico del espectro. La letra *A* es una letra específica del alfabeto.

Un *bit* por definición es un dígito binario. Cuando se usa en asocio con un código binario es mejor pensar que denota una cantidad binaria igual a 0 ó 1. Para representar un grupo de 2^n elementos diferentes en código binario se requiere un mínimo de N bits. Ello es debido a que es posible arreglar r bits en 2^n maneras diferentes. Por ejemplo, un grupo

de cuatro cantidades diferentes puede ser representado por un codigo de dos bits con cada cantidad asignada a cada una de las siguientes combinaciones de bits: 00, 01, 10, 11. Un grupo de ocho elementos requiere un código de tres bits con cada uno de los elementos asignados a uno y sólo uno de los siguientes: 000, 001, 010, 011, 100, 101, 110, 111. Los ejemplos muestran que las diferentes combinaciones en bits de un código de n bits pueden encontrarse contando en forma binaria desde 0 hasta $2^n - 1$. Algunas combinaciones de bits no se asignan cuando el número de elementos de un grupo que va a codificarse no es múltiplo de una potencia de 2. Los diez números decimales 0, 1, 2, . . . , 9 son ejemplos de este grupo. Un código binario que distingue diez elementos diferentes debe contener mínimo cuatro bits; tres bits determinan un máximo de ocho elementos. Cuatro bits pueden conformar 16 combinaciones diferentes, pero como se codifican solamente diez dígitos, las seis combinaciones restantes no se usan ni se asignan.

Aunque el número *mínimo* de bits, necesarios para codificar 2^n cantidades diferentes, es n, no hay un número *máximo* de bits que puedan ser usados por un código binario. Por ejemplo, los diez dígitos decimales pueden ser codificados con diez bits y a cada dígito decimal asignarle una combinación de bits de 9 ceros y un 1. En este código binario en particular, al dígito 6 se le asigna la combinación de bits 0001000000.

Códigos decimales

Los códigos binarios para digitos decimales requieren un mínimo de cuatro bits. Se puede obtener numerosos códigos diferentes rearreglando cuatro o más bits en diez combinaciones posibles. Varias de estas posibilidades se muestran en la Tabla 1-2.

Tabla 1-2 Códigos binarios para dígitos decimales

Dígito decimal	(BDC) 8421	Exceso a 3	84-2-1	2421	(Biguinario) 5043210
0	0000	0011	0000	0000	0100001
1	0001	0100	0111	0001	0100010
2	0010	0101	0110	0010	0100100
3	0011	0110	0101	0011	0101000
4	0100	0111	0100	0100	0110000
5	0101	1000	1011	1011	1000001
6	0110	1001	1010	1100	1000010
7	0111	1010	1001	1101	1000100
8	1000	1011	1000	1110	1001000
9	1001	1100	1111	1111	1010000

El BDC (el binario decimal codificado) es una forma directa asignada a un equivalente binario. Es posible asignar cargas a los bits binarios de acuerdo a sus posiciones. Las cargas en el código BDC son 8, 4, 2, 1. La asignación de bits 0110 por ejemplo, puede ser interpretada por las cargas

para representar el dígito decimal 6 ya que $0\times 8+1\times 4+1\times 2\div 0+1=6$. Es posible asignar cargas negativas a un código decimal, tal como se muestra en el código 8, 4, −2, −1. En este caso la combinación de bits 0110 se interpreta como el dígito decimal 2, al obtenèrse de $0\times 8+1\times 4+1\times(-2)+0\times(-1)=2$. Otros dos códigos con carga mostrados en la tabla son el 2421 y el 5043210. Un código decimal que se ha usado en algunos computadores viejos en el código de exceso a 3. Este último es un código sin carga, cuya asignación se obtiene del correspondiente valor en BDC una vez se haya sumado 3.

Los números se representan en computadores digitales en binario o decimal a través de un código binario. Cuando se estén especificando los datos, el usuario gusta dar los datos en forma decimal. Las maneras decimales recibidas se almacenan internamente en el computador por medio del código decimal. Cada dígito decimal requiere por lo menos cuatro elementos de almacenamiento binario. Los números decimales se convierten a binarios cuando las operaciones aritméticas se hacen internamente con números representados en binario. Es posible también realizar operaciones aritméticas directamente en decimal con todos los números ya dejados en forma codificada. Por ejemplo, el número decimal 395, cuando se convierte a binario da igual a 110001011 y consiste en nueve dígitos binarios. El mismo número representado alternamente en código BDC, ocupa cuatro bits para cada dígito decimal para un total de 12 bits: 001110010101. Los primeros cuatro bits representan el 3, los siguientes cuatro el 9 y los últimos cuatro el 5.

Es muy importante comprender la diferencia entre *conversión* de un número decimal a binario y la *codificación* binaria de un número decimal. En cada caso el resultado final es una serie de bits. Los bits obtenidos de la conversión son dígitos binarios. Los bits obtenidos de la codificación son combinaciones de unos a ceros arregladas de acuerdo a las reglas del código usado. Por tanto es extremadamente importante tener en cuenta, que una serie de unos y ceros en un sistema digital puede algunas veces representar un número binario y otras veces representar alguna otra cantidad discreta de información como se especifica en un código binario dado. El código BDC por ejemplo, ha sido escogido de tal manera que es un código y una conversión binaria directa siempre y cuando los números decimales sean algún entero y entre 0 y 9. Para números mayores que 9, la conversión y la codificación son completamente diferentes. Este concepto es tan importante que vale la pena repetirlo usando otro ejemplo: la conversión binaria del decimal 13 es 1101; la codificación del decimal 13 con BDC es 00010011.

De los cinco códigos binarios listados en la Tabla 1-2, el BDC parece ser el más natural y es sin duda el que se encuentra más comúnmente. Los otros códigos de cuatro bits tienen una característica en común que no se encuentra en BDC. El de exceso a 3, el 2, 4, 2, 1, y el 8, 4, −2, −1 son códigos autocomplementarios, esto es que el complemento de 9 del número decimal se obtiene fácilmente cambiando los más por ceros y los ceros por más. Esta propiedad es muy útil cuando se hacen las operaciones aritméticas interna-

mente con números decimales (en código binario) y la sustracción se hace por medio del complemento de 9.

El código binario mostrado en la Tabla 1-2 es un ejemplo de un código de siete dígitos con propiedades de detección de error. Cada dígito decimal consiste de 5 ceros y 2 unos colocados en las correspondientes columnas de carga.

La propiedad de la detección de error de este código puede comprenderse si uno se da cuenta de que los sistemas digitales representan el binario 1 mediante una señal específica uno y el binario cero por otra segunda señal específica. Durante la trasmisión de señales de un lugar a otro puede presentarse un error. Uno o más bits pueden cambiar de valor. Un circuito en el lado de recepción puede detectar la presencia de más (o menos) de dos unos y en el caso de que la combinación de bits no esté de acuerdo con la combinación permitida, se detectará un error.

Códigos de detección de error

La información binaria, siendo señales de pulsos modulados o señales de entrada y salida de un computador digital, puede ser trasmitida a través de algún medio de comunicación tal como ondas de radio o alambres. Cualquier ruido externo introducido en el medio de comunicación física cambia los valores de los bits de 0 a 1 y viceversa. Puede ser usado un código de detección de error con el objeto de detectar los errores durante la trasmisión. El error detectado no puede ser corregido pero sí indicada su presencia. El procedimiento usual es observar la frecuencia del error. Si el error ocurre de vez en cuando, aleatoriamente y sin algún efecto pronunciado sobre el total de la información trasmitida, o no se hace nada o se trasmite de nuevo el mensaje erróneo específico. Si el error ocurre tan a menudo que se distorciona el significado de la información recibida, se debe rectificar la falla del sistema.

Un bit de *paridad* es un bit extra, incluido con el mensaje para convertir el número total de unos en par o impar. Un mensaje de cuatro bits y un bit de paridad P se representan en la Tabla 1-3. En (a), se escoge P de tal manera que la suma de todos los unos sea impar (en total cinco bits). En (b), se escoge P de tal manera que la suma de todos los unos es par. Durante la trasferencia de información de un lugar a otro, el bit de paridad se trata de la siguiente manera: en el extremo de envío, el mensaje (en el caso de los primeros cuatro bits) se aplica a un circuito "generador de paridad" en el cual se genera el bit P requerido. El mensaje junto con su bit de paridad se trasfiere a su destino. En el extremo de recepción todos los bits entrantes (en este caso cinco) se aplican al circuito de "verificación de paridad' para constatar la paridad adoptada. Se detectará un error si la paridad verificada no corresponde a la adoptada. El método de la paridad detecta la presencia de uno, tres o cualquier combinación de errores impar. Una combinación par de errores no se puede detectar. Una ulterior discusión de la generación de paridad y su verificación puede ser encontrada en la Sección 4-9.

Tabla 1-3 Generación del bit de paridad

(a) Mensaje	P (impar)	(b) Mensaje	P (par)
0000	1	0000	0
0001	0	0001	1
0010	0	0010	1
0011	1	0011	0
0100	0	0100	1
0101	1	0101	0
0110	1	0110	0
0111	0	0111	1
1000	0	1000	1
1001	1	1001	0
1010	1	1010	0
1011	0	1011	1
1100	1	1100	0
1101	0	1101	1
1110	0	1110	1
1111	1	1111	0

El código reflejado

Los sistemas digitales pueden ser diseñados para procesar datos solamente en forma discreta. Muchos sistemas físicos suministran salida continua de datos. Estos datos pueden convertirse en forma discreta o digital antes de ser aplicados a un sistema digital. La información análoga o continua se convierte a forma digital por medio del convertidor análogo a digital. Algunas veces es conveniente usar el código reflejado mostrado en la Tabla 1-4 para representar los datos digitales convertidos en datos análogos. La ventaja del código reflejado sobre los números binarios puros es que el número en el código reflejado cambia en sólo un bit cuando cambia de un número al siguiente. Una aplicación típica del código reflejado ocurre cuando los datos análogos se representan por un cambio continuo de la posición de un eje. El eje se divide en segmentos y a cada segmento se le asigna un número. Si se hace corresponder segmentos adyacentes con números de código reflejados adyacentes, se reduce la ambigüedad cuando se sensa la detección en la línea que separa cualquier par de segmentos. El código reflejado que se muestra en la Tabla 1-4 es solamente uno de los muchos códigos posibles. Para obtener un código reflejado diferente se puede comenzar con cualquier combinación de bits y proceder a obtener la siguiente combinación, cambiando solamente un bit de 0 a 1 ó de 1 a 0 de cualquier modo deseado, al azar, siempre y cuando dos números no tengan códigos asignados idénticos. El código reflejado se conoce como el código *Gray*.

Códigos alfanuméricos

Muchas aplicaciones de computadores digitales, requieren manejar datos que consisten no solamente de números sino también de letras. Por ejem-

Tabla 1-4 Código reflejado de cuatro bits

Código reflejado	Equivalente decimal
0000	0
0001	1
0011	2
0010	3
0110	4
0111	5
0101	6
0100	7
1100	8
1101	9
1111	10
1110	11
1010	12
1011	13
1001	14
1000	15

plo una compañía de seguros con millones de clientes pueden usar un computador digital para procesar sus historias. Para representar el nombre del dueño de una póliza en forma binaria, es necesario tener un código binario para el alfabeto. Además, el mismo código binario puede representar números decimales y algunos otros caracteres especiales. Un código alfanumérico (algunas veces abreviado *alphameric*) es un código binario de un grupo de elementos consistente de los diez números decimales, los 26 caracteres del alfabeto y de cierto número de símbolos especiales tales como $. El número total de elementos de un grupo alfanumérico es mayor que 26. Por consiguiente debe ser codificado con un mínimo de seis bits ($2^6 = 64$, ya que $2^5 = 32$ es insuficiente).

Un arreglo posible de un código alfanumérico de seis bits se muestra en la Tabla 1-5 bajo el nombre de "código interno". Con algunas variaciones se usa en muchas computadoras, para representar internamente caracteres alfanuméricos. La necesidad de representar más de 64 caracteres (las letras minúsculas y los caracteres de control especiales para la trasmisión de información digital) dio lugar a códigos alfanuméricos de siete y ocho bits. Uno de estos códigos es conocido como ASCII (American Standard Code for Information Interchange = Código normalizado americano para el intercambio de información); otro es conocido como EBCDIC (Extended BCD Interchange Code = Código de intercambio BDC aumentado). El código ASCII listado en la Tabla 1-5, consiste de siete bits, pero es para propósitos prácticos un código de ocho bits ya que el octavo bit se agrega de todos modos para efectos de paridad. Cuando se trasfiere información directa mediante tarjetas perforadas, los caracteres alfanuméricos usan un código binario de 12 bits. Una tarjeta perforada consiste en 80 columnas y 12 filas. En cada columna se representa un carácter alfanumérico mediante huecos

Tabla 1-5 Codigos de caracteres alfanuméricos

Caracter	Código interno 6-bits	Código ASCII 7-bits	Código EBCDIC 8-bits	Código de tarjeta 12-bits
A	010 001	100 0001	1100 0001	12,1
B	010 010	100 0010	1100 0010	12,2
C	010 011	100 0011	1100 0011	12,3
D	010 100	100 0100	1100 0100	12,4
E	010 101	100 0101	1100 0101	12,5
F	010 110	100 0110	1100 0110	12,6
G	010 111	100 0111	1100 0111	12,7
H	011 000	100 1000	1100 1000	12,8
I	011 001	100 1001	1100 1001	12,9
J	100 001	100 1010	1101 0001	11,1
K	100 010	100 1011	1101 0010	11,2
L	100 011	100 1100	1101 0011	11,3
M	100 100	100 1101	1101 0100	11,4
N	100 101	100 1110	1101 0101	11,5
O	100 110	100 1111	1101 0110	11,6
P	100 111	101 0000	1101 0111	11,7
Q	101 000	101 0001	1101 1000	11,8
R	101 001	101 0010	1101 1001	11,9
S	110 010	101 0011	1110 0010	0,2
T	110 011	101 0100	1110 0011	0,3
U	110 100	101 0101	1110 0100	0,4
V	110 101	101 0110	1110 0101	0,5
W	110 110	101 0111	1110 0110	0,6
X	110 111	101 1000	1110 0111	0,7
Y	111 000	101 1001	1110 1000	0,8
Z	111 001	101 1010	1110 1001	0,9
0	000 000	011 0000	1111 0000	0
1	000 001	011 0001	1111 0001	1
2	000 010	011 0010	1111 0010	2
3	000 011	011 0011	1111 0011	3
4	000 100	011 0100	1111 0100	4
5	000 101	011 0101	1111 0101	5
6	000 110	011 0110	1111 0110	6
7	000 111	011 0111	1111 0111	7
8	001 000	011 1000	1111 1000	8
9	001 001	011 1001	1111 1001	9
espacio	110 000	010 0000	0100 0000	no perforado
.	011 011	010 1110	0100 1011	12,8,3
(	111 100	010 1000	0100 1101	12,8,5
+	010 000	010 1011	0100 1110	12,8,6
$	101 011	010 0100	0101 1011	11,8,3
*	101 100	010 1010	0101 1100	11,8,4
)	011 100	010 1001	0101 1101	11,8,5
−	100 000	010 1101	0110 0000	11
/	110 001	010 1111	0110 0001	0,1
,	111 011	010 1100	0110 1011	0,8,3
=	001 011	011 1101	0111 1110	8,6

perforados en las columnas adecuadas. Un hueco se sensa como 1 ó su ausencia como 0. Las 12 filas están marcadas, comenzando desde el extremo superior como las filas de perforación 12, 11, 0, 1, 2, . . . , 9. Las tres primeras constituyen el área de perforación de *zona* y las últimas nueve, de perforación *numérica*. El código de tarjeta de 12 bits mostrado en la Tabla 1-5 da un listado de las filas en las cuales se perfora un hueco (dando los unos). Las filas restantes se asumen como ceros. El código de tarjeta de 12 bits es ineficiente con respecto al número de bits con que se usa. La mayoría de los computadores traducen el código de entrada a un código interno de seis bits. Como ejemplo se usa la representación del nombre "John Doe" a continuación:

100001	100110	011000	100101	110000	010100	100110	010101
J	O	H	N	espacio	D	O	E

1-7 ALMACENAMIENTO DE BINARIOS Y REGISTROS

Los elementos discretos de información en un computador digital deben tener una existencia física en algún medio de almacenamiento de información. Además, cuando los elementos discretos de información se representan en forma binaria, el medio de almacenamiento de información debe contener elementos de almacenamiento binario para la acumulación de los bits individuales. Una *celda binaria* es un elemento que posee dos estados estables y es capaz de almacenar un bit de información. La entrada a la celda recibe las señales de exitación que la coloca en uno de los dos estados. La salida de la celda es una cantidad física que distingue entre los dos estados. La información almacenada en la celda es un 1 cuando está en su estado estable y un 0 cuando está en el otro estado estable. Algunos ejemplos de celdas binarias son los circuitos flip-flops, los núcleos de ferrita usados en la memoria y las posiciones perforadas o no de una tarjeta.

Registros

Un *registro* es un grupo de celdas binarias. Como una celda almacena un bit de información, se desprende que un registro de r celdas puede almacenar cualquier cantidad discreta de información que contenga n bits. El *estado* del registro es un número enésimo de unos o ceros con cada bit indicando el estado de una celda en el registro. El *contenido* de un registro es una función de la interpretación dada a la información almacenada en ella. Considérese como ejemplo un registro de 16 celdas:

1	1	0	0	0	0	1	1	1	1	0	0	1	0	0	1
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Físicamente se podría pensar que el registro está compuesto de 16 celdas binarias, con cada celda almacenando un 1 ó un 0. Supongamos que la configuración de bits almacenados es como se muestra en la figura. El estado

del registro es el número 16-avo 1100001111001001. Más claramente, un registro de n celdas puede estar en uno de los 2^n estados posibles. Ahora bien, si se asume que el contenido del registro representa un entero binario, obviamente el registro puede almacenar cualquier número binario de 0 a $2^{16}-1$. Para el caso particular mostrado, el contenido del registro es el equivalente binario al número decimal 50121. Si se asume que el registro almacena caracteres alfanuméricos de un código de 8 bits, el contenido del registro es cualquiera de los caracteres significativos. (Las combinaciones de bits no asignadas no representan información significativa). En el código EBCDIC, el ejemplo anterior representa los 2 caracteres *C* (ocho bits izquierdos) e *I* (ocho bits derechos). Por otra parte, si se interpreta el contenido del registro como cuatro dígitos decimales representados por un código de cuatro bits, el primero será un número decimal de cuatro dígitos. En el código de exceso a 3 del ejemplo anterior se representa el número decimal 9096. En el código BDC el contenido del registro no tiene ningún significado ya que la combinación de bits 1100 no se asigna a ningún dígito decimal. De acuerdo al ejemplo, se nota que un registro puede almacenar uno o más elementos discretos de información y que la misma configuración de bits puede ser interpretada, de manera diferente para diferentes tipos de elementos de información. Es muy importante que el usuario almacene información significativa en registros y que el computador sea programado para procesar esta información de acuerdo al *tipo* de la misma.

Trasferencia entre registros

Un computador digital se caracteriza por sus registros. La unidad de memoria (Figura 1-1) es principalmente una colección de cientos de registros para almacenar información digital. La unidad procesadora se compone de varios registros que almacenan operandos con base en los cuales se realizan operaciones. La unidad de control usa registros para controlar varias secuencias del computador y cada dispositivo de entrada y salida debe tener al menos un registro para almacenar la información trasferida de o al dispositivo. Una operación de *trasferencia entre registros* es una operación básica en sistemas digitales y consiste en la trasferencia de la información almacenada de un registro a otro. La Figura 1-2 ilustra la trasferencia de información entre registros y demuestra pictóricamente la trasferencia de información binaria de un teclado de teletipo a un registro en la unidad de memoria. Se asume que la unidad de entrada del teletipo tiene un teclado, un circuito de control y un registro de entrada. Cada vez que se digita una tecla, el control introduce al registro de entrada un código de carácter alfanumérico equivalente de 8 bits. Se supone que el código usado es el código ASCII con un octavo bit de paridad impar. La información del registro de entrada se trasfiere a las ocho celdas menos significativas del registro procesador. Después de cada trasferencia se borra el registro de entrada para permitir que el control pueda enviar un nuevo código de ocho bits cada vez que se digite el teclado. Cada caracter de ocho bits trasferido al registro procesador viene seguido por un corrimiento del anterior carácter en las siguientes ocho celdas a su iz-

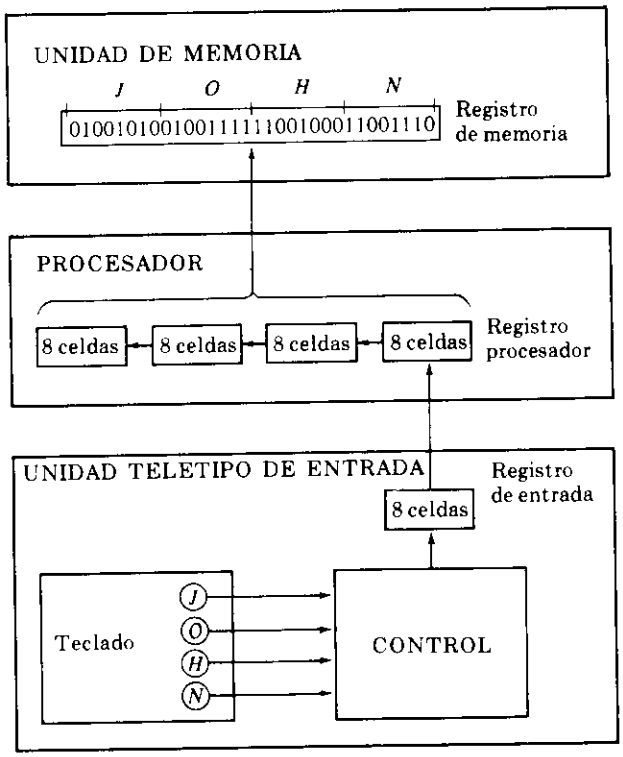


Figura 1-2 Trasferencia de información con registros

quierda. Cuando se complete la trasferencia de cuatro caracteres, el registro procesador estará lleno y su contenido se trasferirá al registro de memoria. El contenido almacenado en el registro de memoria de la Figura 1-2 provino de la trasferencia de los caracteres JOHN después de digitar las cuatro teclas adecuadas.

Para procesar las cantidades discretas de información en forma binaria, el computador debe estar dotado de (1) elementos que sostengan los datos que vayan a ser procesados y (2) elementos de circuito que manejen los bits individuales de información. El elemento más convenientemente usado para retener información es un registro. El manejo de variables binarias se hace por medio de circuitos lógicos digitales. La Figura 1-3 ilustra el proceso de suma de dos números binarios de 10 bits. La unidad de memoria, que consiste usualmente en cientos de registros se muestra en el diagrama con sólo tres de sus registros. La parte de la unidad de proceso mostrada, consiste en tres registros, R1, R2 y R3 conjuntamente con circuitos lógicos digitales que manejan los bits de R1 y R2 y trasfieren a R3 un número binario igual a su suma aritmética. Los registros de memoria almacenan información y están incapacitados para procesar los dos operandos. Sin embargo, la información almacenada en la memoria puede ser trasferida a los registros de proceso. Los resultados obtenidos por el registro del procesador pueden ser trasferidos al registro

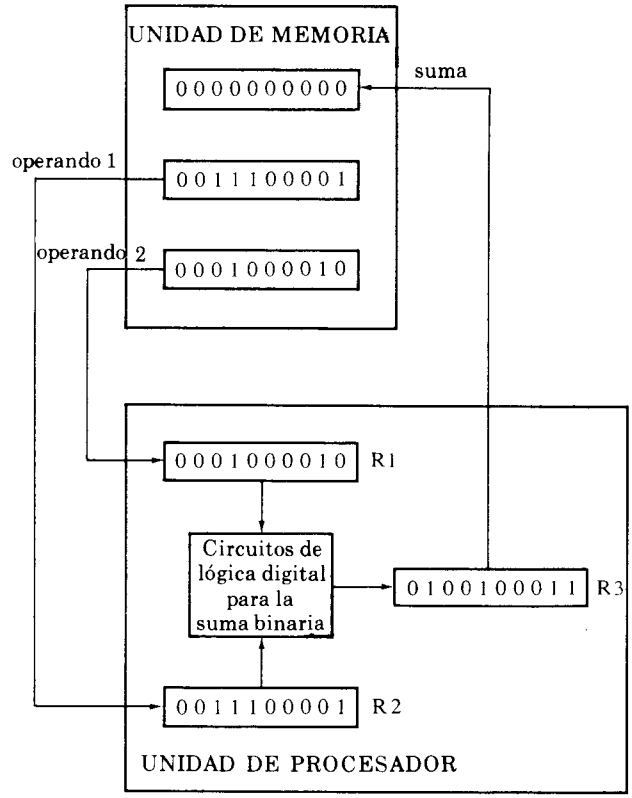


Figura 1-3 Ejemplo de procesamiento de información binaria

de la memoria para almacenamiento hasta que vuelvan a ser necesarios. El diagrama muestra el contenido de los dos operandos trasferidos de los dos registros de memoria R1 y R2. Los circuitos lógicos digitales producen la suma que a su vez será trasferida al registro R3. El contenido del registro R3 puede ser trasladado a los registros de memoria.

Los últimos dos ejemplos demuestran la capacidad del flujo de información del sistema digital de una manera muy sencilla. Los registros del sistema son los elementos básicos para almacenamiento y retención de la información binaria. Los circuitos digitales procesan la información. En la siguiente sección se introducen los circuitos digitales y su correspondiente capacidad de manipulación. El tema de los registros y las operaciones de trasferencia de registros se verá de nuevo en el Capítulo 8.

1-8 LOGICA BINARIA

La lógica binaria trata con variables que toman dos valores discretos y con operaciones que asumen significado lógico. Los dos valores que las variables asumen pueden llamarse de diferentes maneras (por ejemplo, *verdadero* y *falso, si* y *no,* etc.) pero para este propósito es conveniente pensar

en términos de bits y asignar los valores de 1 y 0. La lógica binaria se usa para describir, de una manera matemática el procesamiento y manipuleo de la información binaria. Se acomoda muy bien para el análisis y diseño de los sistemas digitales. Los circuitos lógicos digitales de la Figura 1-3, que realizan la aritmética binaria, son circuitos cuyo comportamiento se expresa más convenientemente en términos de variables binarias y operaciones lógicas. La lógica binaria que se introduce en esta sección es equivalente a un tipo de álgebra llamada álgebra de Boole. La presentación formal del álgebra de Boole de dos valores se verá en más detalles en el Capítulo 2. El propósito de esta sección es el de introducir el álgebra de Boole, de una manera heurística y de relacionarla con los circuitos lógicos digitales y señales binarias.

Definición de lógica binaria

La lógica binaria consiste en variables binarias y operaciones lógicas. Las variables se indentifican mediante las letras del alfabeto tales como *A*, *B*, *C*, *x*, *y*, *z*, etc. y cada variable tendrá dos y sólo dos valores posibles: 1 y 0. Hay tres operaciones lógicas básicas: AND, OR y NOT.

1. AND: Esta operación se representa por un punto o por la ausencia de un operador. Por ejemplo, $x \cdot y = z$ ó $xy = z$ leído "x y y es igual a z" implican que $z = 1$ si y sólo si $x = 1$ y $y = 1$; de otra forma $z = 0$. (Recuérdese que x, y y z son variables y pueden ser solamente 1 ó 0 y nada más.)
2. OR: Esta operación se representa por un signo más. Por ejemplo $x + y = z$ se lee "x OR y es igual a z", queriendo decir que $z = 1$ si $x = 1$ o si $y = 1$ o si se tiene $x = 1$ y $y = 1$. Si ambos $x = 0$ y $y = 0$, entonces $z = 0$.
3. NOT: Esta operación se representa por un apóstrofe (algunas veces por una barra). Por ejemplo: $x' = z$ (ó $\overline{x} = z$) se lee "x no es igual a z" implicando que z es lo que x no. En otras palabras, si $x = 1$ entonces $z = 0$, pero si $x = 0$ entonces $z = 1$.

La lógica aritmética se parece a la aritmética binaria y las operaciones AND y OR tienen su similitud con la multiplicación y la suma respectivamente. De hecho los símbolos usados para AND y OR son los mismos que se usan para la suma y la multiplicación. La lógica binaria, empero, no se debe confundir con la aritmética binaria. Se debe tener en cuenta que una variable aritmética designa un número que puede consistir en muchos dígitos mientras que una variable lógica es siempre 1 ó 0. En la aritmética binaria, por ejemplo, se tiene que $1 + 1 = 10$ (leído "uno más uno es igual a dos") mientras que en la lógica binaria se tiene que $1 + 1 = 1$ (leído: "uno OR uno es igual a uno").

Existe un valor de z especificado por la definición de la operación lógica, por cada combinación de valores x y y. Estas definiciones pueden listarse en una forma compacta usando *tablas de verdad.* Una tabla de verdad es una tabla de todas las combinaciones posibles de las variables

Tabla 1-6 Tablas de verdad de las operaciones lógicas

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

que muestra la relación entre los valores que las variables pueden tomar y el resultado de la operación. Por ejemplo, las tablas de verdad para las operaiones AND y OR con variables x y y se obtienen al listar todos los valores posibles que las variables pueden tener cuando se combinan en pares. El resultado de la operación de cada combinación se lista en una columna separada. Las tablas de verdad de AND, OR y NOT se listan en la Tabla 1-6. Estas tablas demuestran claramente las definiciones de las operaciones.

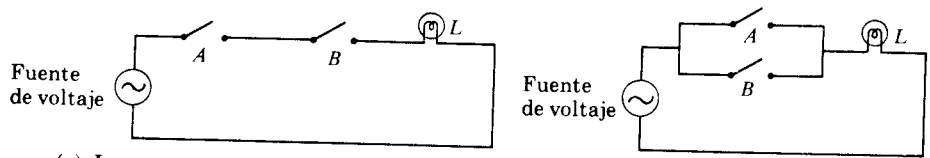
Señales binarias y circuitos de conmutación

El uso de variables binarias y la aplicación a la lógica binaria se demuestra por los circuitos sencillos de conmutación de la Figura 1-4. Supongamos que los interruptores A y B representen dos variables binarias con valores iguales a 0 cuando el interruptor está abierto e igual 1 cuando el interruptor está cerrado. Simultáneamente asúmase que la lámpara L representa una tercera variable primaria igual a 1 cuando la luz está prendida e igual a 0 cuando está apagada. Para el caso de los interruptores en series, la luz se prende solamente si A y B están cerrados. Para los interruptores en paralelo, la luz se prenderá si A o B están cerrados. Obviamente estos dos circuitos pueden expresarse por medio de la lógica binaria con las operaciones AND y OR repectivamente:

$L = A \cdot B$ para el circuito de la Figura 1-4(a)

$L = A + B$ para el circuito de la Figura 1-4(b)

Los circuitos digitales electrónicos se llaman algunas veces *circuitos de conmutación*, ya que se comportan como un interruptor con un elemento activo tal como un transistor conduciendo (interruptor cerrado) o en



(a) Interruptores en serie — AND lógica

(b) Interruptores en paralelo — OR lógico

Figura 1-4 Circuitos de interrupción que demuestran la lógica binaria

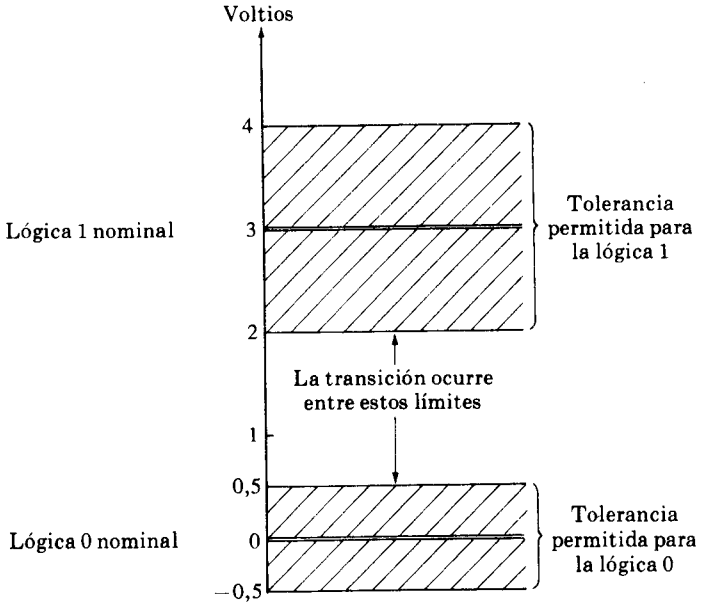
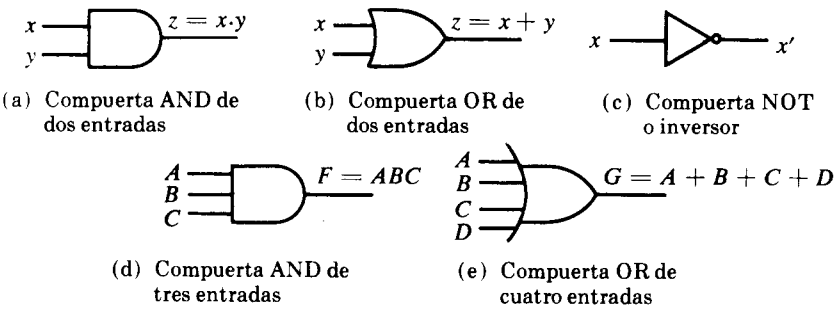


Figura 1-5 Ejemplo de señales binarias

corte (interruptor abierto). En vez de cambiar manualmente el interruptor el circuito de interrupción electrónico usa señales binarias para controlar el estado de conducción o no conducción del elemento activo. Las señales eléctricas tales como voltajes o corrientes existen por todo el sistema digital en cualquiera de los dos valores reconocibles (excepto durante la transición). Los circuitos operados por voltaje responden a dos niveles separados los cuales representan una variable binaria igual a lógica 1 o lógica 0. Un sistema digital en particular podría definir la lógica 1 como una señal de valor nominal de 3 voltios y la lógica 0 como una señal de valor nominal de 0 voltios. Como se muestra en la Figura 1-5 cada nivel de voltaje tiene una desviación aceptable de la nominal. La región intermedia entre las regiones permitidas se cruza solamente durante las transiciones de estado. Los terminales de entrada de los circuitos digitales aceptan señales binarias dentro de las tolerancias permisibles y responden en el terminal de salida con señales binarias que caen dentro de las tolerancias específicas.

Compuertas lógicas

Los circuitos digitales electrónicos se llaman *circuitos lógicos* ya que con las entradas adecuadas establecen caminos de manipuleo lógico. Cualquier información deseada para calcular o controlar, puede ser operada pasando señales binarias a través de varias combinaciones de circuitos lógicos con cada señal que representa una variable y trasporta un bit de información. Los circuitos lógicos que ejecutan las operaciones lógicas de AND, OR y NOT se muestran con sus respectivos símbolos en la Figura 1-6.



(a) Compuerta AND de dos entradas

(b) Compuerta OR de dos entradas

(c) Compuerta NOT o inversor

(d) Compuerta AND de tres entradas

(e) Compuerta OR de cuatro entradas

Figura 1-6 Símbolos para los circuitos lógicos

Estos circuitos, llamados *compuertas* son bloques de circuitería que producen señales de salida de lógica 1 o lógica 0, si se satisfacen las condiciones de las entradas lógicas. Nótese que se han usado cuatro nombres diferentes para el mismo tipo de circuito: circuitos digitales, circuitos de conmutación, circuitos lógicos y compuertas. Todos los cuatro nombres se usan a menudo pero se hará referencia a los circuitos como compuertas AND, OR y NOT. La compuerta NOT se denomina algunas veces como *circuito inversor* ya que invierte la señal binaria.

Las señales de entrada x y y en las compuertas de dos entradas de la Figura 1-6 pueden existir en uno de los cuatro estados posibles: 00, 10, 11 ó 01. Estas señales de entrada se muestran en la Figura 1-7 conjuntamente con las señales de salida de las compuertas AND y OR. Los diagramas de tiempo de la Figura 1-7 ilustran la respuesta de cada circuito a cada una de las posibles combinaciones binarias de entrada. La razón para el nombre "inversor" dado a la compuerta NOT es aparente al comparar la señal x (entrada del inversor) y la señal x' (salida del inversor).

Las compuertas AND y OR, pueden tener más de dos entradas como la compuerta AND con tres entradas y la compuerta OR con cuatro entradas de la Figura 1-6. La compuerta AND de tres entradas responde con la salida de lógica 1 si todas las tres señales de entrada son de lógica 1. La salida produce una señal de lógica 0 si cualquier entrada es de lógica 0. La compuerta 0 de cuatro entradas responde con lógica 1 cuando cualquier entrada es de lógica 1. Su salida será de lógica 0 si todas las señales de entrada son de lógica 0.

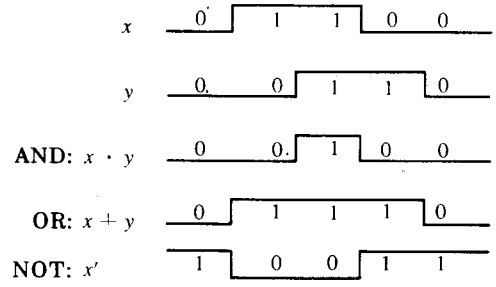


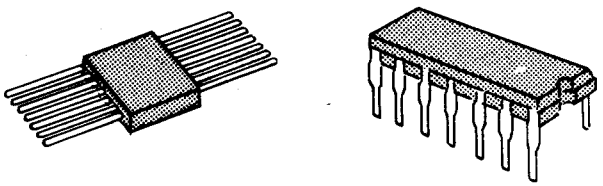
Figura 1-7 Señales de entrada-salida para las compuertas (a), (b) y (c) de la Figura 1-6

El sistema matemático de lógica binaria es mejor conocido como de Boole o álgebra de conmutación. Esta álgebra se usa convenientemente para describir la operación de conjuntos complejos de circuitos digitales. Los diseñadores de los sistemas digitales usan el álgebra de Boole para trasformar los diagramas de circuito a expresiones algebraicas o viceversa. Los Capítulos 2 y 3 se dedican al estudio del álgebra de Boole, sus propiedades y su capacidad de manipuleo. El Capítulo 4 muestra cómo el álgebra de Boole puede usarse para expresar matemáticamente las interconexiones entre los enlaces de compuertas.

1-9 CIRCUITOS INTEGRADOS

Los circuitos digitales están construidos invariablemente con circuitos integrados. Un circuito integrado (abreviado CI) es un cristal semiconductor de silicón, llamado *pastilla,* que contiene componentes eléctricos tales como transistores, diodos, resistencias y condensadores. Los diversos componentes están interconectados dentro de la pastilla para formar un circuito electrónico. La pastilla está montada en un empaque plástico con sus conexiones soldadas a las patillas externas para conformar el circuito integrado. Los circuitos integrados difieren de otros circuitos electrónicos compuestos de elementos discretos en que los componentes individuales del CI no pueden ser separados o desconectados y que el circuito dentro del paquete se hace accesible solamente por medio de las patillas externas.

Los circuitos integrados vienen en dos clases de pastillas, la pastilla plana y la pastilla de hilera doble de patillas* tal como se ve en la Figura 1-8. La pastilla de hilera doble es la más comúnmente usada debido a su bajo costo y fácil instalación en los circuitos impresos. La protección del circuito integrado se hace de plástico o cerámica. La mayoría de las pastillas tienen tamaños normalizados y el número de patillas varían entre 8 y 64. Cada circuito integrado tiene su designación numérica impresa sobre su superficie, para poder identificarlo. Cada fabricante publica un libro de características o catálogo para suministrar la información correspondiente a los diversos productos.



Pastilla plana

Pastilla de hilera doble de patillas

Figura 1-8 Circuitos integrados

*En inglés se usa (DIP) Dual-in-line package.

El tamaño del circuito integrado es bastante pequeño. Por ejemplo, cuatro compuertas AND están escapsuladas dentro de una pastilla de 14 patillas en hilera doble con dimensiones de $20 \times 8 \times 3$ milímetros. Un microprocesador completo está encapsulado de una pastilla de 40 patillas en hilera doble con dimensiones de $50 \times 15 \times 4$ milímetros.

Además de la reducción sustancial de tamaño el CI ofrece otras ventajas y beneficios comparados con los circuitos electrónicos con componentes discretos. El costo de los CI es bastante bajo, lo cual los hace económicos de usarlos. Su bajo consumo de poder hace los sistemas digitales más económicos de operar. Tienen una gran confiabilidad de no fallar y por tanto menos reparaciones. La velocidad de operación es alta haciéndolos más adecuados para operaciones de alta velocidad. El uso de los CI reduce el número de conexiones externas ya que la mayoría están internamente dentro de la pastilla. Debido a todas estas ventajas, los sistemas digitales se construyen con circuitos integrados.

Los circuitos integrados se clasifican en dos categorías generales: *lineales* y *digitales*. Los CI lineales operan con señales continuas para producir funciones electrónicas tales como amplificadores y comparadores de voltaje. Los circuitos integrados digitales, operan con señales binarias y se hacen de compuertas digitales interconectadas. Aquí se tratará solamente con los circuitos integrados digitales.

A medida que mejora la tecnología de los CI, el número de compuertas que pueden encapsularse en una pastilla de silicón, ha aumentado considerablemente. La forma de diferenciar aquellos CI que tengan unas pocas compuertas, con las que tienen cientos de compuertas, es referirse a la pastilla como un elemento de integración pequeña, mediana o grande. Unas pocas compuertas en una sola pastilla constituyen un elemento de integración pequeña (SSI).* Para poder calificar como un elemento de integración mediana (MSI)* el circuito integrado debe cumplir una función lógica completa y tener una complejidad de 10 a 100 compuertas. Un elemento de integración a gran escala (LSI)* realiza una función lógica con más de 100 compuertas. Existe también una integración de muy grande escala (VLSI)* para aquellos elementos que contienen miles de compuertas en una sola pastilla.

Muchos diagramas de circuitos digitales considerados en este libro, se muestran en detalle hasta describir las compuertas individuales y sus interconexiones. Tales diagramas son útiles para demostrar la construcción lógica de una función particular. Sin embargo, debemos tener en cuenta en la práctica que una función dada se obtiene de un elemento de mediana o gran integración (MSI y LSI), al cual el usuario sólo tiene acceso a las entradas externas o salidas pero nunca a las entradas o salidas de las compuertas intermedias. Por ejemplo, un diseñador que desee incorporar un registro en su sistema debe preferiblemente escoger tal función de un circuito de mediana integración (MSI), en vez de diseñar los circuitos digitales individuales como se muestra en el diagrama.

*En inglés se usa: SSI (Small scale integration) Integración de pequeña escala; MSI (Medium scale integration) Integración de mediana escala; LSI (Large scale integration) Integración a gran escala; VLSI (Very large scale integration) Integración a muy grande escala.

REFERENCIAS

1. Richard, R. K., *Arithmetic Operations in Digital Computers*. Nueva York: Van Nostrand Co., 1955.
2. Flores, I., *The Logic of Computer Arithmetic*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.
3. Chu, Y., *Digital Computer Design Fundamentals*. Nueva York: McGraw-Hill Book Co., 1962, Capítulos 1 y 2.
4. Kostopoulos, G. K., *Digital Engineering*. Nueva York: John Wiley & Sons, Inc., 1975, Capítulo 1.
5. Rhyne, V. T., *Fundamentals of Digital Systems Design*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1973, Capítulo 1.

PROBLEMAS

1-1. Escriba los primeros 20 dígitos decimales en base 3.

1-2. Sume y multiplique los siguientes números en la base dada sin convertirlos a decimal.

(a) $(1230)_4$ y $(23)_4$
(b) $(135,4)_6$ y $(43,2)_6$
(c) $(367)_8$ y $(715)_8$
(d) $(296)_{12}$ y $(57)_{12}$

1-3. Convierta el número decimal 250,5 a base 3, 4, 7, 8 y 16 respectivamente.

1-4. Convierta los siguientes números decimales a binarios: 12,0625, 10^4, 673,23 y 1.998.

1-5. Convierta los siguientes binarios a decimales:
10,10001, 101110,0101, 1110101,110, 1101101,111.

1-6. Convierta los siguientes números en base a las bases que se indican:

(a) El decimal 225,225 a binario, octal y hexadecimal.
(b) El binario 11010111,110 a decimal, octal y hexadecimal.
(c) El octal 623,77 a decimal, binario y hexadecimal.
(d) El hexadecimal 2AC5,D a decimal, octal y binario.

1-7. Convierta los siguientes números a decimal:

(a) $(1001001,011)_2$
(b) $(12121)_3$
(c) $(1032,2)_4$
(d) $(4310)_5$
(e) $(0,342)_6$
(f) $(50)_7$
(g) $(8,3)_9$
(h) $(198)_{12}$

1-8. Obtenga el complemento de 1 y de 2 de los siguientes números binarios:
1010101, 0111000, 0000001, 10000, 00000.

1-9. Obtenga el complemento de 9 y de 10 de los siguientes números decimales:
13579, 09900, 90090, 10000, 00000.

1-10. Encuentre el complemento de 10 de $(935)_{11}$.

1-11. Haga la sustracción de los números decimales a continuación, usando (1) el complemento de 10 (2) el complemento de 9. Compruebe la respuesta por medio de la resta directa.

(a) $5250 - 321$
(b) $3570 - 2100$
(c) $753 - 864$
(d) $20 - 1000$

1-12. Realice la sustracción, de los siguientes números binarios usando (1) el complemento de 2 (2) el complemento de 1. Compruebe la respuesta por sustracción directa.

(a) $11010 - 1101$
(b) $11010 - 10000$
(c) $10010 - 10011$
(d) $100 - 110000$

1-13. Pruebe el procedimiento expuesto en la Sección 1-5 para la sustracción de dos números con complemento de $(r-1)$.

1-14. Para los códigos cargados (a) 3, 3, 2, 1 y (b) 4, 4, 3, -2 para números decimales, determine todas las tablas posibles de tal manera que el complemento de 9 de cada dígito decimal se obtenga mediante el cambio de unos a ceros y de ceros a unos.

1-15. Represente el número decimal 8620 (a) en BDC, (b) en código de exceso 3, (c) el código 2, 4, 2, 1 y (d) como número binario.

1-16. Un código binario usa diez bits para representar cada uno de los diez dígitos decimales. A cada dígito se le asigna un código de nueve ceros y un 1. El código binario para 6, por ejemplo, es 0001000000. Determine el código binario para los dígitos decimales restantes.

1-17. Obtenga el código binario cargado para los dígitos de base 12 usando las cargas de 5421.

1-18. Determine el bit de paridad impar generado cuando el mensaje consiste en diez dígitos decimales en el código 8, 4, -2, -1.

1-19. Determine otras dos combinaciones distintas al código reflejado mostrado en la Tabla 1-4.

1-20. Obtenga un código binario para representar todos los dígitos en base 6 de tal manera que el complemento de 5 se obtenga remplazando 1 por 0 y 0 por 1 en cada uno de los bits del código.

1-21. Asigne un código binario de alguna manera ordenada a las 52 cartas de la baraja. Se debe usar el menor número de bits.

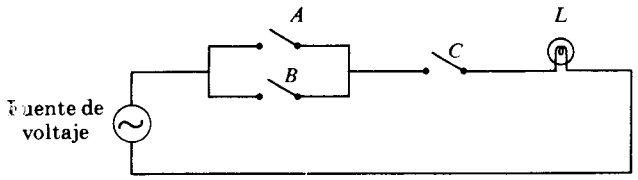
1-22. Escriba su nombre y apellidos en un código de ocho bits compuesto de los siete bits ASCII de la Tabla 1-5 y un bit de paridad par localizado en la posición más significativa. Incluya los espacios entre las partes del nombre y el punto después de la inicial del segundo apellido.

1-23. Muestre la configuración de un registro de 24 celdas cuando su contenido representa (a) el número $(295)_{10}$ en binario, (b) el número decimal 295 en BDC y (c) los caracteres XY5 en EBCDIC.

1-24. El estado de un registro de 12 celdas es 010110010111. ¿Qué significa su contenido si este representa (a) tres dígitos decimales en BDC, (b) tres dígitos decimales en código de exceso 3, (c) tres dígitos decimales en código 2, 4, 2, 1 y (d) dos caracteres en el código interno de la Tabla 1-5?

1-25. Muestre el contenido de todos los registros en la Figura 1-3 si los dos números binarios agregados tienen el equivalente decimal de 257 y 1050. Asuma un registro con 8 celdas.

1-26. Exprese el siguiente circuito de conmutación en notación lógica binaria.



1-27. Muestre las señales (usando un diagrama similar al de la Figura 1-7) de las salidas *F* y *G* de la Figura 1-6. Use señales arbitrarias en las entradas *A*, *B*, *C* y *D*.

Algebra de Boole y compuertas lógicas

2

2-1 DEFINICIONES LOGICAS

El álgebra de Boole, como cualquier otro sistema matemático deductivo puede ser definida por un conjunto de elementos, un conjunto de operadores, un número de axiomas o postulados. Un *conjunto* de elementos es una colección de objetos que tienen una propiedad común. Si S es un conjunto y x y y son objetos ciertos, entonces $x \in S$ denota que x es un miembro del conjunto S y $y \notin S$ denota que y no es un elemento de S. Un conjunto con un número finito de elementos se representa por medio de llaves: $A = \{1, 2, 3, 4\}$, es decir los elementos del conjunto A son los números 1, 2, 3 y 4. Un *operador binario* definido en un conjunto S de elementos, es una regla que asigna a cada par de elementos de S un elemento único de S. Por ejemplo, considérese la relación $a * b = c$. Se dice que $*$ es un operador binario si éste especifica una regla para encontrar c de un par (a, b) y también si $a, b, c \in S$. Por otra parte, $*$ no es un operador binario si $a, b \in S$ mientras que la regla encuentre que $c \notin S$.

Los postulados de un sistema matemático forman las suposiciones de las cuales se deducen las reglas, teorías y propiedades del mismo. Los postulados más comúnmente usados para formular varias extructuras algebraicas son:

1. *Conjunto cerrado.* Un conjunto S es cerrado con respecto a un operador binario, si para cada par de elementos de S, el operador binario especifica una regla para obtener un elemento único de S. El conjunto de los números naturales $N = \{1, 2, 3, 4, \ldots\}$, por ejemplo, es cerrado con respecto al operador binario $(+)$ por las reglas de la suma aritmética ya que por cada $a, b \in N$ se obtiene una $c \in N$ única por la operación $a + b = c$. El conjunto de los números naturales no es cerrado con respecto al operador binario menos $(-)$ por las reglas de la sustracción aritmética ya que $2 - 3 = -1$ y $2, 3 \in N$ mientras que $(-1) \notin N$.
2. *Ley asociativa.* Se dice que un operador binario $*$ en un conjunto S es asociativo si:

$$(x*y)*z = x*(y*z) \quad \text{para toda } x, y, z \in S$$

3. *Ley conmutativa.* Se dice que un operador binario $*$ en un conjunto S es conmutativo si:

$$x*y = y*x \quad \text{para toda } x, y \in S$$

4. *Elemento de identidad.* Se dice que un conjunto S tiene un elemento de identidad con respecto a la operación binaria $*$ en S si existe un elemento $e \in S$ con la propiedad:

$$e*x = x*e = x \quad \text{para toda } x \in S$$

Ejemplo: El elemento 0 es un elemento de identidad con respecto a la operación $+$ en el conjunto de enteros $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$ ya que:

$$x + 0 = 0 + x = x \text{ para toda } x \in I$$

El conjunto de números naturales N no tiene elemento de identidad ya que el 0 es excluido del mismo.

5. *Inverso.* Se dice que un conjunto S, que tiene un elemento de identidad e con respecto a un operador binario $*$, tiene un inverso si para cada $x \in S$ existe un elemento $y \in S$ tal que:

$$x*y = e$$

Ejemplo: En el conjunto de enteros I con $e = 0$, el inverso del elemento a es $(-a)$ ya que $a + (-a) = 0$.

6. *Ley distributiva.* Si $*$ y $\cdot$ son dos operadores binarios en un conjunto S, se dice que $*$ es distributivo con respecto a $\cdot$ si:

$$x*(y \cdot z) = (x*y) \cdot (x*z)$$

Un ejemplo de una extructura algebraica es un *campo*. Un campo es un conjunto de elementos agrupados con dos operadores binarios, cada uno de los cuales tiene las propiedades 1 a 5 que se combinan para dar la propiedad 6. El conjunto de números reales conjuntamente con los operadores binarios $+$ y $\cdot$ forman el campo de los números reales. El campo de los números reales es la base de la aritmética y el álgebra ordinaria. Los operadores y postulados tienen los siguientes significados:

El operador binario $+$ define la suma.

La identidad aditiva es 0.

El inverso aditivo define la sustracción.

El operador binario $\cdot$ define la multiplicación.

La identidad multiplicativa es 1.

El inverso multiplicativo de $a = 1/a$ define la división, es decir, $a \cdot 1/a = 1$.

La única ley distributiva aplicable es la de $\cdot$ sobre $+$:

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

2-2 DEFINICION AXIOMATICA DEL ALGEBRA BOOLEANA

En 1854 George Boole (1) introdujo un tratamiento sistemático de lógica y para ello desarrolló un sistema algebraico que hoy en día llamamos *álgebra de Boole.* En 1938 C. E. Shannon (2) introdujo una álgebra de Boole de dos valores llamada *álgebra de conmutación* en la cual él demostró que las propiedades de los circuitos de conmutación eléctricas biestables pueden ser representadas por esta álgebra. Se usarán los postulados formulados por E. V. Huntington (3) en 1904 para la definición formal del álgebra de Boole. Estos postulados y axiomas no son únicos para definir el álgebra de Boole ya que se ha usado otro conjunto de postulados. *El álgebra de Boole es una estructura algebraica definida para un conjunto de elementos B juntamente con dos operadores binarios $+$ y $\cdot$, de tal forma que se satisfagan los siguientes postulados (Huntington):

1. (a) Conjunto cerrado con respecto al operador $+$.
   (b) Conjunto cerrado con respecto al operador $\cdot$.
2. (a) Un elemento de identidad con respecto a $+$ designado por el 0: $x + 0 = 0 + x = x$.
   (b) Un elemento de identidad con respecto a $\cdot$ designado por 1: $x \cdot 1 = 1 \cdot x = x$.
3. (a) Conmutativo con respecto a $+$: $x + y = y + x$.
   (b) Conmutativo con respecto a $\cdot$: $x \cdot y = y \cdot x$.
4. (a) $\cdot$ es distributivo sobre $+$: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (b) $+$ es distributivo sobre $\cdot$: $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. Para cada elemento $x \in B$, existe un elemento $x' \in B$ (llamado el complemento de x) tal que: (a) $x + x' = 1$ y (b) $x \cdot x' = 0$.
6. Existen al menos dos elementos x, $y \in B$ tales que $x \neq y$.

Al comparar el álgebra de Boole con la aritmética y el álgebra ordinaria (el de los números reales) se notan las siguientes diferencias:

1. Los postulados de Huntington no incluyen la ley asociativa. Sin embargo esta ley es válida para el álgebra de Boole y puede deducirse (para muchos operadores) de otros postulados.
2. La ley distributiva de $+$ sobre $\cdot$, es decir, $x + (y \cdot z) = (x + y) \cdot (x + z)$ es válida para el álgebra de Boole pero no para el álgebra ordinaria.
3. El álgebra de Boole no tiene inversos aditivos o multiplicativos y por tanto no hay operaciones de sustracción o división.
4. El postulado 5 define un operador llamado *complemento* el cual no está disponible en el álgebra ordinaria.

*Ver por ejemplo Birkoff y Bartee (4), Capítulo 5.

5. El álgebra ordinaria trata con los números reales, los cuales constituyen un conjunto infinito de elementos. El álgebra de Boole trata con los elementos B hasta ahora no definidos pero que se definen a continuación para el álgebra de Boole de dos valores (de mucho interés para el uso ulterior de esta álgebra), B está definido como un conjunto de solamente dos elementos, 0 y 1.

El álgebra Boole se asemeja al álgebra ordinaria en algunos aspectos. La escogencia de los símbolos + y · es intencional con el fin de facilitar las manipulaciones con álgebra de Boole por parte de personas familiarizadas con el álgebra ordinaria. Aunque no se puede usar algunos conocimientos derivados del álgebra ordinaria para tratar con álgebra de Boole, el principiante debe ser muy cuidadoso de no sustituir las reglas del álgebra ordinaria donde no sean aplicables.

Es muy importante distinguir entre los elementos del conjunto de una estructura algebraica y las variables de un sistema algebraico. Por ejemplo, los elementos del campo de los números reales son números mientras que las variables tales como a, b, c, etc., usadas en el álgebra ordinaria son símbolos que se establecen para los números reales. Similarmente en el álgebra de Boole se definen los elementos de un conjunto B y las variables, tales que x, y, z sean simplemente símbolos que representen los elementos. A estas alturas es importante darse cuenta que para tener una álgebra de Boole se debe demostrar:

1. los elementos del conjunto B,
2. las reglas de operación de los dos operadores binarios, y
3. que el conjunto de elementos B, juntamente con los dos operadores, satisfaga los seis postulados de Huntington.

Se pueden formular muchas álgebras de Boole dependiendo de la escogencia de los elementos de B y las reglas de operación.* En el trabajo subsiguiente, se tratará solamente con una álgebra de Boole bivalente, es decir, una con dos elementos. El álgebra de Boole bivalente tiene aplicaciones en la teoría de conjuntos (el álgebra de enseñanza) y en la lógica de proposiciones. El interés en este libro es en la aplicación del álgebra de Boole a los circuitos con compuertas.

Algebra booleana bivalente

Una álgebra de Boole bivalente se define sobre un conjunto de dos elementos $B = \{0, 1\}$, con reglas para los operadores binarios + y · de la manera como se muestra en las siguientes tablas de operador. (La regla para el operador complemento es para verificación del postulado 5):

Estas reglas son exactamente las mismas que las operaciones AND, OR y NOT respectivamente y que se han definido en la Tabla 1-6. Se debe demos-

*Ver por ejemplo, Hohn (6) Whitesitt (7), o Birkhoff y Bartee (4).

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

trar que los postulados Huntington son válidos para el conjunto $B = \{0, 1\}$ y para los dos operadores binarios definidos anteriormente.

1. *El conjunto cerrado* es obvio a partir de las tablas ya que el resultado de cada operación es 1 ó 0 y 1, $0 \in B$.
2. De las tablas se observa que:

   (a) $0 + 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$
   (b) $1 \cdot 1 = 1 \qquad 1 \cdot 0 = 0 \cdot 1 = 0$

   lo cual establece los dos *elementos de identidad* 0 para $+$ y 1 para $\cdot$ de la manera como se definen en el postulado 2.
3. Las leyes *conmutativas* son obvias de la simetría de las tablas de los operadores binarios.
4. (a) La ley *distributiva* $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$, puede demostrarse que es verdadera de las tablas del operador, al formar la tabla de verdad de todos los valores posibles de x, y y z. Para cada combinación se puede derivar $x \cdot (y + z)$ y demostrar que ese valor es el mismo que $(x \cdot y) + (x \cdot z)$.

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

   (b) La ley *distributiva* de $+$ sobre $\cdot$ puede demostrarse que es verdadera, mediante una tabla de verdad similar a la descrita anteriormente.
5. De la tabla de complementos se puede demostrar fácilmente que:
   (a) $x + x' = 1$, ya que $0 + 0' = 0 + 1 = 1$ y $1 + 1' = 1 + 0 = 1$
   (b) $x \cdot x' = 0$, ya que $0 \cdot 0' = 0 \cdot 1 = 0$ y $1 \cdot 1' = 1 \cdot 0 = 0$ lo cual verifica el postulado 5.

6. El postulado 6 se satisface, ya que el álgebra bivalente tiene dos elementos distintos 1 y 0 con $1 \neq 0$.

Se ha establecido una álgebra de Boole bivalente que tiene un conjunto de dos elementos 1 y 0, dos operadores binarios con reglas de operación equivalentes a las operaciones AND y OR y el operador complemento equivalente al operador NOT. Así, el álgebra de Boole ha sido definida de una manera matemática formal y se ha demostrado que es equivalente a la lógica binaria representada heurísticamente en la Sección 1-8. La representación heurística es una ayuda para entender la aplicación del álgebra de Boole a los circuitos tipo compuertas. La representación formal es necesaria para desarrollar los teoremas y propiedades del sistema algebraico. El álgebra de Boole bivalente definida en esta sección, es llamada por los ingenieros "álgebra de conmutación". Para darle énfasis a la similitud que hay entre el álgebra de Boole bivalente y otros sistemas binarios, se le ha llamado "lógica binaria" en la Sección 1-8. De aquí en adelante se omitirá el adjetivo bivalente del álgebra de Boole en las discusiones subsiguientes.

2-3 TEOREMAS BASICOS Y PROPIEDADES DEL ALGEBRA BOOLEANA

Dualidad

Los postulados de Huntington han sido listados en pares y repartidos en parte (a) y parte (b). Una parte puede obtenerse de otra si los operadores binarios y los elementos de identidad son intercambiables. Este principio importante del álgebra de Boole se llama el *principio de dualidad.* Este último establece que las expresiones algebraicas deducidas de los postulados del álgebra de Boole permanecen válidos si se intercambian los operadores y elementos de identidad. En el álgebra de Boole bivalente, los elementos de identidad y los elementos del conjunto B son los mismos: 1 y 0. El principio de dualidad tiene muchas aplicaciones. Si se desea una expresión algebraica *dual*, se intercambia simplemente los operadores OR y AND y se remplaza unos por ceros y ceros por unos.

Teoremas básicos

En la Tabla 2-1 se listan los seis teoremas del álgebra de Boole y cuatro de sus postulados. La notación se simplifica omitiendo el · toda vez que no cause confusión. Los teoremas y postulados listados son las relaciones más básicas en el álgebra de Boole. Se advierte al lector que debe familiarizarse con ellas tan pronto como pueda. Tanto los teoremas como los postulados se listan en pares y cada relación es dual con la que está apareada. Los postulados son axiomas básicos de la extructura algebraica y no necesitan prueba. Los teoremas deben probarse a partir de los postulados. Las pruebas de los teoremas con una variable se presentan a continuación. En la parte derecha se lista el número del postulado que justifica cada paso de la prueba.

Tabla 2-1 Postulados y teoremas del álgebra de Boole

Postulado 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulado 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Teorema 1	(a) $x + x = x$	(b) $x \cdot x = x$
Teorema 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Teorema 3, involución	$(x')' = x$	
Postulado 3, conmutativo	(a) $x + y = y + x$	(b) $xy = yx$
Teorema 4, asociativo	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulado 4, distributivo	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Teorema 5, DeMorgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Teorema 6, absorción	(a) $x + xy = x$	(b) $x(x + y) = x$

TEOREMA 1(a): $x + x = x$.

$x + x = (x + x) \cdot 1$	del postulado: 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

TEOREMA 1(b): $x \cdot x = x$.

$x \cdot x = xx + 0$	del postulado: 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Nótese que el teorema 1(b) es el dual del teorema 1(a) y que cada paso de la prueba en parte (b) es el dual de la parte (a). Cualquier teorema dual puede derivarse similarmente de la prueba de un par correspondiente.

TEOREMA 2(a): $x + 1 = 1$.

$x + 1 = 1 \cdot (x + 1)$	del postulado: 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

TEOREMA 2(b): $x \cdot 0 = 0$ por dualidad.

TEOREMA 3: $(x')' = x$. Del postulado 5, se tiene $x + x' = 1$ y $x \cdot x' = 0$, lo cual define el complemento de x. El complemento de x' es x y es también $(x')'$. Así como el complemento es único tendremos que $(x')' = x$.

Los teoremas que comprenden dos o tres variables pueden ser probados algebraicamente de los postulados y de los teoremas ya probados. Tómese por ejemplo el teorema de absorción.

TEOREMA 6(a): $x + xy = x$.

$$
\begin{aligned}
x + xy &= x \cdot 1 + xy && \text{del postulado 2(b)} \\
&= x(1 + y) && \text{del postulado 4(a)} \\
&= x(y + 1) && \text{del postulado 3(a)} \\
&= x \cdot 1 && \text{del teorema 2(a)} \\
&= x && \text{del postulado 2(b)}
\end{aligned}
$$

TEOREMA 6(b): $x(x + y) = x$ por dualidad.

Los teoremas del álgebra de Boole pueden demostrarse por medio de las tablas de verdad. En estas tablas, ambos lados de la relación se comprueban para arrojar resultados idénticos para todas las combinaciones posibles de los variables integrantes. La siguiente tabla de verdad verifica el primer teorema de absorción.

			=
x	y	xy	$x + xy$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Las pruebas algebraicas de la ley asociativa y del teorema de De Morgan son largas y no se dará una prueba de ellas. Sin embargo, su validez es fácilmente demostrable mediante las tablas de verdad. Por ejemplo, la tabla de verdad para el primer teorema de De Morgan $(x + y)' = x'y'$ se muestra a continuación:

x	y	$x + y$	$(x + y)'$	x'	y'	$x'y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Prioridad del operador

La prioridad del operador para la evaluación de las expresiones de Boole es (1) el paréntesis, (2) NOT, (3) AND y (4) OR. En otras palabras las expresiones dentro de un paréntesis deben ser evaluadas antes de otras operaciones. La siguiente operación en orden prioritario es el complemento, luego sigue la AND y finalmente la OR. Como ejemplo, considérese la tabla de verdad del teorema de De Morgan. El lado izquierdo de la expresión es

$(x+y)'$. Así, la expresión dentro del paréntesis es evaluada primero y luego se complementa el resultado. El lado derecho de la expresión es $x'y'$. Por tanto, el complemento de x y el complemento de y se evalúan primero y el resultado se somete a una operación AND. Nótese que en la aritmética se tiene en cuenta la misma prioridad (excepto para el complemento) cuando la multiplicación y la suma se remplazan por AND y OR respectivamente.

Diagrama de Venn

Una figura útil que puede ser usada para visualizar las relaciones entre las variables del álgebra de Boole es el *diagrama de Venn*. Este diagrama consiste en un rectángulo tal como el que se muestra en la Figura 2-1, en el cual se dibujan círculos traslapados para cada una de las variables. Cada círculo es designado por una variable. Se asignan todos los puntos dentro del círculo como pertenecientes a dichas variables y todos los puntos por fuera del círculo como no pertenecientes a la variable. Tómese por ejemplo el círculo designado x. Si estamos dentro del círculo, se dice que $x=1$ y cuando estamos fuera de él se dice que $x=0$. Ahora bien, con dos círculos traslapados se forman cuatro áreas distintas dentro del rectángulo: el área que no pertenece ni a x ni a y $(x'y')$, el área dentro del círculo y pero por fuera de x $(x'y)$, el área dentro del círculo y pero por fuera de y (xy') y el área dentro de ambos círculos (xy).

Los diagramas de Venn se usan para demostrar los postulados del álgebra de Boole y para demostrar la validez de los teoremas. La Figura 2-2, por ejemplo, muestra que el área que pertenece a xy está dentro del círculo x y por tanto $x+xy=x$. La Figura 2-3 ilustra la ley distributiva $x\ (y+z)=xy+xz$. En este diagrama se tienen tres círculos traslapados para cada una de las variables x, y y z. Es posible distinguir ocho áreas diferentes en el diagrama de Venn de tres variables. Para este ejemplo en particular, se demuestra la ley distributiva al notar que el área de

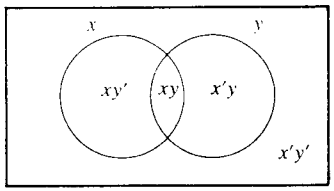


Figura 2-1 Diagrama de Venn de dos variables

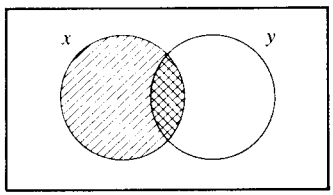
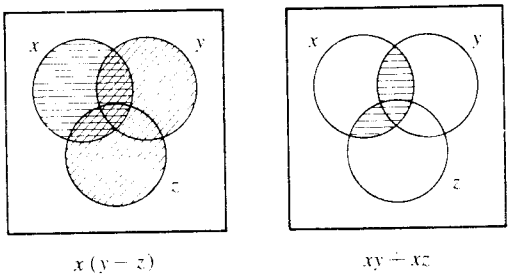


Figura 2-2 Ilustración del diagrama de Venn $x=xy+x$



$x(y+z)$ $\qquad$ $xy+xz$

Figura 2-3 Ilustración del diagrama de Venn para la ley distributiva

intersección entre el círculo x con el área que contiene y ó z es la misma área que pertenece a xy ó xz.

2-4 FUNCIONES BOOLEANAS

Una variable binaria puede tomar el valor 0 ó 1. Una función de Boole es una expresión formada con variables binarias, dos operadores binarios OR y AND, el operador NOT, el paréntesis y el signo igual. Para un valor dado de variables, la función puede ser 0 ó 1. Considérese por ejemplo la función de Boole:

$$F_1 = xyz'$$

La función F_1 es igual a 1 si $x = 1$ y $y = 1$ y $z' = 1$; de otra manera $F_1 = 0$. El ejemplo anterior es una función de Boole representada como una expresión algebraica. Una función de Boole puede ser representada por medio de una tabla de verdad. Para hacerlo se necesita una lista de 2^n combinaciones de unos y ceros de las n variables binarias y una columna mostrando las combinaciones para las cuales la función es igual a 1 ó 0. Como se muestra en la Tabla 2-2 existen ocho posibles combinaciones diferentes para asignar bits en las tres variables. La columna demarcada F_1 contiene un 0 ó un 1 para cada una de estas combinaciones. La Tabla muestra que la función F_1 es igual a 1 solamente cuando $x = 1$, $y = 1$ y $z = 0$. Para cualquier otra combinación $F_1 = 0$. (Nótese que la afirmación $z' = 1$ es equivalente a decir que $z = 0$.) Considérese la siguiente función:

$$F_2 = x + y'z$$

$F_2 = 1$ si $x = 1$ ó si $y = 0$, mientas $z = 1$. En la Tabla 2-2, $x = 1$ en las últimas cuatro filas y $yz = 01$ en las filas 001 y 101. La última combinación se aplica también para $x = 1$. Así, hay cinco combinaciones para hacer $F_2 = 1$. Como tercer ejemplo, considérese la función:

$$F_3 = x'y'z + x'yz + xy'$$

Esto se muestra en la Tabla 2-2 con cuatro unos y cuatro ceros. F_4 es lo mismo que F_3 y se considera a continuación:

Tabla 2-2 Tablas de verdad para $F_1 = xyz'$, $F_2 = x + y'z$, $F_3 = x'y'z + x'yz + xy'$, y $F_4 = xy' + x'z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Cualquier función de Boole puede ser representada por una tabla de verdad. El número de filas en la tabla es de 2^n donde n es el número de variables binarias de la función. Las combinaciones de unos y ceros se pueden obtener fácilmente para cada fila de los números binarios contando desde 0 a $2^n - 1$. Para cada fila de la tabla, hay un valor para la función igual a 1 ó 0. Se formula ahora la pregunta: ¿Hay una expresión algebraica única para una función de Boole dada? En otras palabras: ¿Es posible encontrar dos expresiones algebraicas para especificar la misma función? La respuesta para estas preguntas es sí. De hecho, la manipulación del álgebra de Boole se aplica mayormente al problema de encontrar expresiones más simples para la misma función. Considérese por ejemplo la función:

$$F_4 = xy' + x'z$$

De la Tabla 2-2 se encuentra que F_4 es idéntica a F_3, ya que ambas tienen unos y ceros idénticos para cada combinación de valores de las tres variables binarias. En general, dos funciones de n variables binarias son iguales si ellas tienen el mismo valor para todas las 2^n combinaciones posibles de las n variables.

Una función de Boole puede ser trasformada de una expresión algebraica a un diagrama lógico compuesto de compuertas AND, OR y NOT. La realización de las cuatro funciones introducidas en la anterior discusión se muestra en la Figura 2-4. Los diagramas lógicos incluyen un circuito inversor para cada variable presente en su forma de complemento. (El inversor no es necesario si se cuenta con el complemento de la variable.) Hay una compuerta AND para cada término de la expresión y una compuerta OR para combinar dos o más términos. De los diagramas, es obvio que para completar F_4 se requieren menos compuertas y entradas que F_3. Como F_4 y F_3 son funciones de Boole iguales, es más económico llevar a cabo la forma F_4 que la forma F_3. Para encontrar circuitos más sencillos, se debe conocer cómo manipular las funciones de Boole para obtener funciones iguales pero simplificadas. Lo que constituye la mejor forma de una expresión de Boole, depende de la aplicación particular. En esta sección se considera el criterio de minimización de equipo.

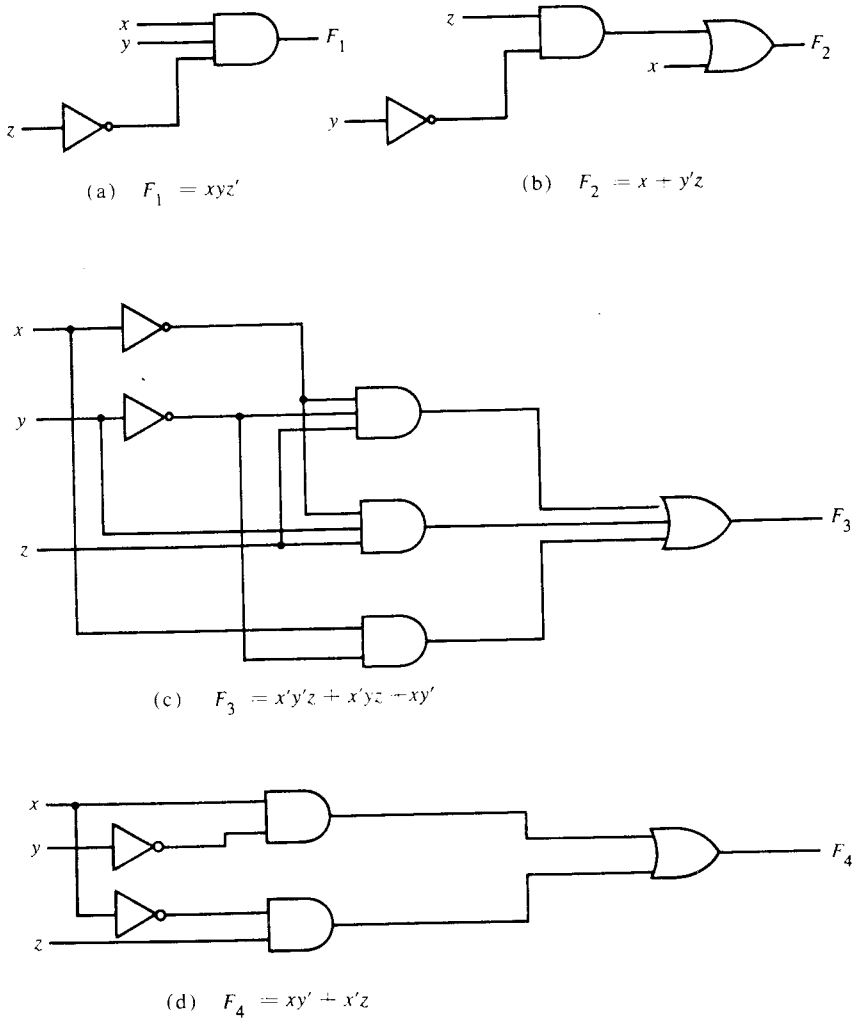


Figura 2-4 Ejecución de las funciones de Boole con compuertas

Manipulación algebraica

Un *literal* es una variable tildada o no tildada. Cuando una función de Boole se ejecuta con compuertas lógicas, cada literal o letra de la función designa una entrada a cada compuerta y cada término se realiza con una compuerta. La minimización del número de literales y el número de términos dará como resultado un circuito con menos componentes. No es siempre posible minimizar ambos simultáneamente. Por lo regular se tienen disponibles otros criterios. Por el momento se limitará el criterio de minimización a la minimización de literales. *Posteriormente* se discutirán otros criterios en el Capítulo 5. El número de literales en una función de Boole puede ser minimizado por medio de manipulaciones algebraicas.

Desafortunadamente no hay reglas específicas a seguir que garanticen una respuesta final. El único método disponible es el procedimiento "tratar y acortar" usando los postulados, los teoremas básicos y cualesquier otros métodos de manipulación que se hagan familiares con el uso. Los siguientes ejemplos ilustran este procedimiento.

EJEMPLO 2-1: Simplifíquese la siguiente función de Boole al mínimo número de literales.

1. $x + x'y = (x + x')(x + y) = 1 \cdot (x + y) = x + y$
2. $x(x' + y) = xx' + xy = 0 + xy = xy$
3. $x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$
4. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z \end{aligned}$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$ por dualidad de la función 4.

Las funciones 1 y 2 son duales entre sí y usan expresiones duales en los pasos correspondientes. La función 3 muestra la igualdad de las funciones F_3 y F_4 tratadas anteriormente. La cuarta demuestra que un aumento en el número de literales, algunas veces, produce una expresión final más simple. La función 5 no se minimiza directamente pero puede deducirse de la dual de los pasos usados para deducir la función 4.

Complemento de una función

El complemento de la función F es F' y se obtiene del intercambio de ceros a unos y unos a ceros en el valor de F. El complemento de una función puede derivarse algebraicamente del teorema de De Morgan. Este par de teoremas están listados en la Tabla 2-1 para dos variables. Los teoremas de De Morgan pueden extenderse a tres o más variables. La forma de tres variables del primer teorema de De Morgan se deriva a continuación. Los postulados y los teoremas son aquellos listados en la Tabla 2-1.

$(A + B + C)' = (A + X)'$	hágase $B + C = X$
$= A'X'$	del teorema 5(a) (De Morgan)
$= A' \cdot (B + C)'$	sustitúyase $B + C = X$
$= A' \cdot (B'C')$	del teorema 5(a) (De Morgan)
$= A'B'C'$	del teorema 4(b) (asociativo)

Los teoremas de De Morgan para cualquier número de variables se parecen al caso de las dos variables y pueden derivarse por sustituciones sucesivas similares al método usado en la derivación hecha anteriormente. Estos teoremas pueden generalizarse de la siguiente manera:

$$(A + B + C + D + \cdots + F)' = A'B'C'D' \ldots F'$$
$$(ABCD \cdots F)' = A' + B' + C' + D' + \ldots + F'$$

La forma generalizada del teorema de De Morgan expresa que el complemento de una función se obtiene intercambiando los operadores AND y OR y complementando cada literal.

EJEMPLO 2-2: Encuéntrese el complemento de las funciones $F_1 = x'yz' + x'y'z$ y $F_2 = x(y'z' + yz)$. Aplicando el teorema de De Morgan tantas veces como sea necesario se obtienen los complementos de la siguiente manera:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')' \cdot (yz)'$$
$$= x' + (y + z)(y' + z')$$

Un procedimiento más sencillo para derivar el complemento de una función es tomando el dual, de una función y complementando cada literal. Este método se deduce del teorema de De Morgan generalizado. Se debe recordar que el dual de cada función se obtiene intercambiando los operadores AND y OR y los unos y ceros.

EJEMPLO 2-3: Encontrar el complemento de la función F_1 y F_2 del Ejemplo 2-2 tomando los duales y complementando cada literal.

1. $F_1 = x'yz' + x'y'z$.
   El dual de F_1 es $(x' + y + z')(x' + y' + z)$.
   Complementando cada literal: $(x + y' + z)(x + y + z') = F_1'$.
2. $F_2 = x(y'z' + yz)$.
   El dual de F_2 es $x + (y' + z')(y + z)$.
   Complementando cada literal: $x' + (y + z)(y' + z') = F_2'$.

2-5 FORMAS CANONICA Y NORMALIZADA

Términos mínimos y términos máximos

Una variable binaria puede aparecer en su forma normal (x) o en la forma de complemento (x'). Considérese ahora dos variables binarias x y y combinadas con la operación AND; como cada variable puede aparecer de cualquier forma, habrá cuatro combinaciones posibles: $x'y'$, $x'y$, xy' y xy. Cada uno de estos cuatro términos AND representan una de las diferentes áreas del diagrama de Venn de la Figura 2-1 y se llaman *términos mínimos* (minterm) de un producto normalizado. De igual manera, se pueden cambiar n variables para formar 2^n términos mínimos. Los 2^n diferentes términos mínimos pueden determinarse por un método similar al mostrado en

Tabla 2-3 Términos mínimos y máximos para tres variables binarias

			Términos mínimos		Términos máximos	
x	y	z	Término	Designación	Término	Designación
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

la Tabla 2-3 para tres variables. Los números binarios de 0 a $2^n - 1$ se listan bajo las n variables. Cada término mínimo se obtiene de un término AND de n variables con cada variable tildada, si el bit correspondiente al número binario es 0 y si no está tildada a 1. Un símbolo para cada término mínimo se ilustra en la tabla en la forma de m_j, donde j denota, el equivalente decimal del número binario del término mínimo correspondiente.

De manera similar, las n variables formando un término OR, con cada variable tildada o no tildada, darán 2^n combinaciones posibles llamadas *términos máximos* (maxterms) de las *sumas normalizadas*. Los ocho términos máximos de las tres variables, conjuntamente con la simbología asignada, se listan en la Tabla 2-3. Cualesquier 2^n términos para n variables pueden determinarse de manera similar. Cada término máximo se obtiene de un término OR de n variables con cada variable no tildada si el correspondiente bit es 0 y tildada si es 1.* Nótese que cada término máximo es el complemento de su correspondiente término mínimo y viceversa.

Una función de Boole puede ser expresada algebraicamente a partir de una tabla de verdad dada, conformando un término mínimo por cada combinación de las variables que producen un 1 en la función para luego obtener la OR de todos los términos. Por ejemplo, la función en la Tabla 2-4 se determina expresando las combinaciones 001, 100, 111 como $x'y'z$, $xy'z'$ y x y z respectivamente. Como cada uno de estos términos mínimos resulta en $f_1 = 1$, se tiene:

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

De manera similar, se puede fácilmente verificar que:

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

*Algunos textos definen un término máximo (maxterms) como un término OR de n variables con cada variable no tildada si el bit es 1 y tildada si es 0. La definición adoptada en este libroes preferible ya que lleva a conversiones más normales entre las funciones tipo término máximo y término mínimo.

Tabla 2-4 Funciones de tres variables

x	y	z	Función f_1	Función f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Estos ejemplos demuestran una propiedad importante del álgebra de Boole. Cualquier función de Boole puede ser expresada como una suma de términos mínimos (por "suma" se quiere decir la suma OR de los términos).

Considérese ahora el complemento de una función de Boole. Este puede leerse de una tabla de verdad formando un término mínimo por cada combinación que produce un cero y luego haciendo la función OR de esos términos. El complemento de f_1 se lee así:

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Si se obtiene el complemento de f_1' se obtiene la función f_1:

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

De igual manera, es posible leer la expresión f_2 de la tabla:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

Estos ejemplos demuestran una segunda propiedad importante del álgebra de Boole: cualquier función de Boole puede expresarse como un producto de términos máximos (por "producto" se implica el producto AND de los términos). El procedimiento para obtener el producto de términos máximos directamente de una tabla de verdad se logra de la siguiente manera: fórmese un término máximo para cada combinación de variables que produzcan un 0 en la función y luego forme la función AND de todos los términos máximos. A las funciones de Boole expresadas como una suma de términos mínimos o producto de términos máximos se les dice que están en *forma canónica*.

Suma de términos mínimos

Se había dicho antes que para n variables binarias, se pueden obtener 2^n términos mínimos diferentes y que cualquier función de Boole puede

expresarse como una suma de términos mínimos. Los términos mínimos cuya suma define la función de Boole son aquellos que dan el 1 de la función en una tabla de verdad. Como la función puede ser 1 ó 0 para cada término mínimo y ya que hay 2^n términos mínimos, se pueden calcular las funciones posibles que pueden formarse con n variables como 2^{2^n}. Algunas veces es conveniente expresar la función de Boole en la forma de suma de términos mínimos. Si no está en esta forma, se puede llegar a ella expandiendo primero la expresión a una suma de términos AND. Luego se inspecciona cada término para ver si contiene todas las variables. Si le hace falta una o más variables, se aplica la función AND con una expresión tal como $x + x'$, donde x sea una de las variables faltantes. El siguiente ejemplo aclara este procedimiento.

EJEMPLO 2-4: Expresar la función de Boole $F = A + B'C$ como suma de términos mínimos. La función tiene tres variables: A, B y C. Como el primer término A no tiene las otras dos variables por tanto:

$$A = A(B + B') = AB + AB'$$

Como la expresión carece de una variable:

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

El segundo término $B'C$ carece también de una variable:

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Combinando todos los términos se obtendrá:

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + AB'C + A'B'C \end{aligned}$$

Pero como $AB'C$ aparece dos veces, y de acuerdo al teorema 1 $(x + x = x)$, es posible quitar uno de ellos. Rearreglando los términos en orden ascendente se obtendrá finalmente:

$$\begin{aligned} F &= A'B'C + AB'C' + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

Es conveniente algunas veces, expresar la función de Boole cuando está compuesta de una suma de términos mínimos por medio de la siguiente forma simplificada:

$$F(A, B, C) = \Sigma\,(1, 4, 5, 6, 7)$$

El símbolo de sumatoria $\sum$ implica los términos a los cuales se les aplica la función OR. Los términos entre paréntesis son los términos míni-

mos de la función. Las letras entre paréntesis a continuación de la F forman la lista de las variables en el orden tomado cuando el término mínimo se convierte en un término AND.

Producto de términos máximos

Cada una de las 2^{2^n} funciones de n variables binarias pueden expresarse como un producto de términos máximos. Para expresar las funciones de Boole como un producto de términos máximos se debe primero llevar a una forma de términos OR. Esto puede lograrse usando la ley distributiva $x + yz = (x + y)(x + z)$ y si hay una variable x faltante en cada término OR se le aplicará la función OR conjuntamente con xx'. Este procedimiento se clarifica por medio del siguiente ejemplo:

EJEMPLO 2-5: Expresar la función de Boole $F = xy + x'z$ como un producto en la forma de términos máximos. Primero conviértase la función a términos OR usando la ley distributiva:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

La función tiene tres variables: x, y y z. A cada término OR le hace falta una variable, por tanto:

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combinando todos los términos y quitando aquellos que aparezcan más de una vez se obtendrá finalmente:

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

Una forma conveniente de expresar esta función es de la siguiente manera:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

El símbolo de producto Π denota la aplicación de la función AND a los términos máximos. Los números representan los términos máximos de la función.

Conversión entre las formas canónicas

El complemento de una función expresada como la suma de términos mínimos es igual a la suma de los términos mínimos faltantes de la función original. Esto último es debido a que la función original es expresada por

aquellos términos mínimos que hacen la función igual a 1 mientras que un complemento es un 1 para aquellos términos mínimos en que la función es un 0. Como ejemplo considérese la función:

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Esta función tiene un complemento que puede expresarse así:

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Ahora si se obtiene el complemento de F' por el teorema de De Morgan obtendremos una F de manera diferente:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

La última definición se deriva de la definición de los términos mínimos y términos máximos que figuran en la Tabla 2-3. De la tabla, es claro que es válida la siguiente relación:

$$m_j' = M_j$$

Esto es, el término máximo con suscrito j es un complemento de un término mínimo con el mismo suscrito j y viceversa.

El último ejemplo demuestra la conversión entre una función expresada como una suma de términos mínimos a su equivalente como producto de términos máximos. Con un argumento similar se mostrará que la conversión entre el producto de términos máximos y la suma de los términos mínimos es similar. Se establece ahora un procedimiento de conversión general. Para hacer la conversión de una forma canónica a otra, intercámbiese los símbolos $\sum$ y Π y lístese aquellos números que faltan en la forma original. Como otro ejemplo, la función:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

se expresa como producto de la forma de términos máximos. Su conversión a la suma de términos mínimos será:

$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

Nótese que para poder encontrar los términos faltantes, se debe tener en cuenta que el número total de términos mínimos y términos máximos es 2^n en donde n es el número variable binario en la función.

Formas normalizadas

Las dos formas del álgebra de Boole son formas básicas que se obtienen al leer la función de la tabla de verdad. Estas formas muy raramente son las que tienen el menor número de literales debido a que cada término mínimo o término máximo, debe contener por definición, *todas* las variables complementadas o no.

Otra forma de expresar las funciones de Boole es la forma *normalizada*. En esta configuración, los términos que forman la función deben con-

tener uno, dos o cualquier número de literales. Hay dos tipos de formas normalizadas: la suma de productos y el producto de sumas.

La *suma de productos* es una expresión de Boole que contiene términos AND llamados *términos producto* de uno o más literales cada uno. La *suma* denota la aplicación de la función OR de estos términos. Un ejemplo de una función expresada en suma de productos es:

$$F_1 = y' + xy + x'yz'$$

Esta expresión tiene tres términos producto de uno, dos y tres literales cada uno, respectivamente. Su suma es en efecto una operación OR.

Un *producto de sumas* es una expresión de Boole que contiene términos OR, llamados *términos suma*. Cada término puede tener cualquier número de literales. El *producto* denota la aplicación de la función AND a estos términos. Un ejemplo de una expresión en producto de sumas es:

$$F_2 = x(y' + z)(x' + y + z' + w)$$

La expresión tiene tres términos suma de uno, dos y cuatro literales cada uno. El producto es una operación AND. El uso de las palabras *producto* y *suma* se establece debido a la similitud de la operación AND y el producto aritmético (multiplicación) y la similitud de la operación OR con la suma aritmética (adición).

Una función de Boole puede ser expresada en una forma no normalizada. Por ejemplo la función:

$$F_3 = (AB + CD)(A'B' + C'D')$$

no es ni suma de productos ni producto de sumas. Puede cambiarse a una forma normalizada usando la ley distributiva para quitar el paréntesis:

$$F_3 = A'B'CD + ABC'D'$$

2-6 OTRAS OPERACIONES LOGICAS

Cuando los operadores binarios AND y OR se colocan entre las dos variables x y y, ellas forman las funciones de Boole $x \cdot y$ y $x + y$, respectivamente. Se estableció previamente que hay 2^{2^n} funciones de n variables binarias. Para dos variables, $n = 2$ el número de funciones de Boole posibles es 16. Por tanto las funciones AND y OR son solamente dos del total de las 16 funciones posibles formadas con dos variables primarias. Sería muy instructivo encontrar las otras 14 funciones e investigar sus propiedades.

Las tablas de verdad para las 16 funciones formadas con dos variables binarias x y y, se listan en la Tabla 2-5. En esta tabla, cada una de las 16 columnas F_0 a F_{15} representan una tabla de verdad de una función posible para las dos variables dadas x y y. Nótese que las funciones se determinan a partir de las 16 combinaciones binarias, que pueden ser asignadas a F. Algunas de las funciones se muestran con un símbolo operador. Por ejemplo, F_1 representa la tabla de verdad para una AND y F_7 represen-

Tabla 2-5 Tablas de verdad para las 16 funciones de dos variables binarias

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Símbolo operador			.	/		/		$\oplus$	+	$\downarrow$	$\odot$	′	$\subset$	′	$\supset$	$\uparrow$	

ta la tabla de verdad para la OR. Los símbolos operadores para estas funciones son (·) y (+) respectivamente.

Las 16 funciones listadas en la tabla de verdad pueden ser expresadas algebraicamente por medio de expresiones de Boole. Esto se puede ver en la primera columna de la Tabla 2-6. Las expresiones de Boole listadas están simplificadas al mínimo número de literales.

Aunque cada función puede ser expresada en términos de los operadores de Boole AND, OR y NOT, no hay razón para no poder asignar símbolos operadores especiales para expresar las otras funciones. Tales símbolos operadores se listan en la segunda columna de la Tabla 2-6. Sin embargo,

Tabla 2-6 Expresiones de Boole para 16 funciones de dos variables

Funciones de Boole	Símbolo operador	Nombre	Comentarios
$F_0 = 0$		Nulo	Constante binaria 0
$F_1 = xy$	$x \cdot y$	AND	x y y
$F_2 = xy'$	x/y	Inhibición	x pero no y
$F_3 = x$		Trasferencia	x
$F_4 = x'y$	y/x	Inhibición	y pero no x
$F_5 = y$		Trasferencia	y
$F_6 = xy' + x'y$	$x \oplus y$	OR-exclusiva	x ó y pero no ambas
$F_7 = x + y$	$x + y$	OR	x ó y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	No-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalencia*	x igual a y
$F_{10} = y'$	y'	Complemento	No y
$F_{11} = x + y'$	$x \subset y$	Implicación	Si y entonces x
$F_{12} = x'$	x'	Complemento	No x
$F_{13} = x' + y$	$x \supset y$	Implicación	Si x entonces y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	No-AND
$F_{15} = 1$		Identidad	Constante binaria 1

**Equivalencia* es conocida también como *igualdad coincidencia* y *NOR exclusiva.*

todos los símbolos nuevos mostrados, con excepción del símbolo de la OR-exclusiva $\oplus$, no son de uso común por parte de los diseñadores digitales.

Cada una de las funciones en la Tabla 2-6 se lista con su correspondiente nombre y comentario que explica su función de forma simple. Las 16 funciones listadas pueden subdividirse en tres categorías:

1. Dos funciones que producen una constante 0 ó 1.
2. Cuatro funciones con operaciones unarias de complemento y trasferencia.
3. Diez funciones con operadores binarios que definen ocho operaciones diferentes AND, OR, NAND, NOR, OR-exclusiva, equivalencia, inhibición e implicación.

Cualquier función puede ser igual a una constante, pero una función binaria puede ser igual solamente a 1 ó 0. La función complemento produce el complemento de cada una de las variables. A la función que es igual a la variable de entrada se le ha dado el nombre de *trasferencia* ya que la variable x ó y es trasferida a través de compuertas que forman la función sin cambiar su valor. De los ocho operadores binarios, dos (inhibición e implicación) son usados por los logistas, pero muy rara vez se usan en lógica de computadores. Los operadores AND y OR se han mencionado conjuntamente con el álgebra de Boole. Las otras cuatro funciones se usan mucho en el diseño de sistemas digitales.

La función NOR es el complemento de la función OR y su nombre es una contracción de *not-OR*. De manera similar, NAND es el complemento de AND y es una contracción de *not-AND*. La OR-exclusiva, abreviado XOR ó EOR es similar al OR pero excluye la combinación de *ambos* x y y igual a 1. La equivalencia es una función que es 1 cuando las dos variables son iguales, es decir, cuando ambas son cero o ambas son 1. La OR-exclusiva y la función de equivalencia son complementarias entre sí. Esto puede ser verificado fácilmente al inspeccionar la Tabla 2-5. La tabla de verdad para la OR-exclusiva es F_6 y para la equivalencia es F_9 y estas dos funciones se complementan entre sí. Por esta razón la función de equivalencia se llama a menudo NOR-exclusiva, es decir OR-exclusiva NOT.

El álgebra de Boole tal como se ha definido en la Sección 2-2, tiene dos operadores binarios que nosotros hemos llamado AND y OR y el operador unario NOT (complemento). De las definiciones, se ha deducido un número de propiedades de estos operadores y se han definido ahora otros operadores binarios en términos de los primeros. No hay nada especial acerca de este procedimiento. Se hubiera podido comenzar con el operador NOR ($\downarrow$), por ejemplo, para posteriormente definir AND, OR y NOT en términos del primero. No obstante, estas son buenas razones para introducir el álgebra de Boole de la forma que se ha hecho. Los conceptos "and", "or" y "not" son familiares y la gente los usa día a día para expresar ideas lógicas. Además, los postulados de Huntington reflejan la naturaleza doble del álgebra haciendo énfasis en la simetría de $+$ y $\cdot$ entre sí.

2-7 COMPUERTAS LOGICAS DIGITALES

Como las funciones de Boole se expresan en términos de operaciones AND, OR y NOT, es más fácil llevar a cabo una función de Boole con este tipo de compuertas. La posibilidad de construir compuertas para las otras operaciones lógicas es de interés práctico. Los factores que van a ser valorizados cuando se considera la construcción de otros tipos de compuertas lógicas son (1) la factibilidad y economía de producir la compuerta con compuertas físicas, (2) la posibilidad de expandir la compuerta a más de dos entradas, (3) las propiedades básicas del operador binario tales como conmutatividad y asociatividad y (4) la habilidad de la compuerta para llevar a cabo las funciones de Boole por sí solas o conjuntamente con otras.

De las 16 funciones definidas en la Tabla 2-6, dos son iguales a una constante y las otras cuatro se repiten dos veces. Quedan solamente diez funciones para ser consideradas como candidatas para compuertas lógicas. Dos de ellas, la inhibición e implicación no son conmutativas o asociativas y por tanto imprácticas de usar como compuertas lógicas normalizadas. Las otras ocho: complemento, trasferencia, AND, OR, NAND, NOR, OR-exclusiva y equivalencia se usan como compuertas normalizadas para el diseño digital.

Los símbolos gráficos y las tablas de verdad de las ocho compuertas se muestran en la Figura 2-5. Cada compuerta tiene una o dos entradas variables designadas como x y y y una variable de salida binaria designada como F. Los circuitos AND, OR e inversor fueron definidos en la Figura 1-6. El circuito inversor invierte el sentido lógico de una variable binaria y produce la función NOT o complemento. El círculo pequeño a la salida del símbolo gráfico de un inversor implica un complemento lógico. El símbolo triángulo designa para sí solo un circuito separador (buffer). Un circuito separador produce la función de *trasferencia* pero no produce ninguna operación lógica particular ya que el valor binario de la salida es igual al valor binario de la entrada. Este circuito se usa solamente para amplificación de señal de potencia y es equivalente a dos inversores conectados en cascada.

La función NAND es el complemento de la función AND tal como se indica por el símbolo gráfico que consiste en un símbolo gráfico AND seguido de un pequeño círculo. La función NOR es el complemento de la función OR y usa un símbolo gráfico OR seguido de un pequeño círculo. Las compuertas NAND y NOR se usan mucho como compuertas lógicas normalizadas y de hecho son más populares que las compuertas AND y OR. Ello se debe a que las compuertas NAND y NOR pueden construirse fácilmente con transistores y además porque las funciones de Boole pueden llevarse a cabo fácilmente con ellas.

La compuerta OR-exclusiva tiene un símbolo gráfico similar al de la compuerta OR excepto por una línea curva adicional del lado de la entrada. La equivalencia o compuerta NOR-exclusiva es el complemento de la OR-exclusiva de la manera como indica un pequeño círculo del lado de la salida del símbolo gráfico.

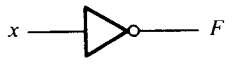
Nombre	Símbolo gráfico	Función algebraica	Tabla de verdad
AND		$F = xy$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$
OR		$F = x + y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 1$
Inversor		$F = x'$	$x \mid F$ $0 \mid 1$ $1 \mid 0$
Separador		$F = x$	$x \mid F$ $0 \mid 0$ $1 \mid 1$
NAND		$F = (xy)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR		$F = (x + y)'$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 0$
OR-exclusiva (XOR)		$F = xy' + x'y$ $= x \oplus y$	$x\ y \mid F$ $0\ 0 \mid 0$ $0\ 1 \mid 1$ $1\ 0 \mid 1$ $1\ 1 \mid 0$
NOR-exclusiva o equivalencia		$F = xy + x'y'$ $= x \odot y$	$x\ y \mid F$ $0\ 0 \mid 1$ $0\ 1 \mid 0$ $1\ 0 \mid 0$ $1\ 1 \mid 1$

Figura 2-5 Compuertas lógicas digitales

Expansión a entradas múltiples

Las compuertas mostradas en la Figura 2-5 a excepción del inversor y el separador pueden expandirse a más de dos entradas. Una compuerta puede expandirse a múltiples entradas si la operación binaria que representa es conmutativa y asociativa. Las operaciones AND y OR definidas en el álgebra de Boole tienen estas dos propiedades. Para la función OR se tiene:

$$x + y = y + x \qquad \text{conmutativo}$$

y

$$(x + y) + z = x + (y + z) = x + y + z \qquad \text{asociativo}$$

lo cual indica que las compuertas de entrada pueden intercambiarse y que la función OR puede extenderse a tres o más variables.

Las funciones NAND y NOR son conmutativas y sus compuertas pueden expandirse para más de dos entradas si se tiene en cuenta que la operación se modifica un poco. La dificultad es que los operadores NAND y NOR no son asociativos, es decir, $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$, como se ve a continuación:

$$(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$$
$$x \downarrow (y \downarrow z) = [x + (y + z)']' = x'(y + z) = x'y + x'z$$

Para vencer esta dificultad, se define una compuerta NOR múltiple (ó NAND) como una OR complementada (ó AND). Así, por definición se tiene:

$$x \downarrow y \downarrow z = (x + y + z)'$$
$$x \uparrow y \uparrow z = (xyz)'$$

Los símbolos gráficos de las compuertas de tres entradas se muestran en la Figura 2-7. Al escribir operaciones con NOR y NAND en cascada se debe tener en cuenta el correcto uso del paréntesis para implicar la secuencia adecuada de las compuertas. Para demostrar lo anterior considérese el cir-

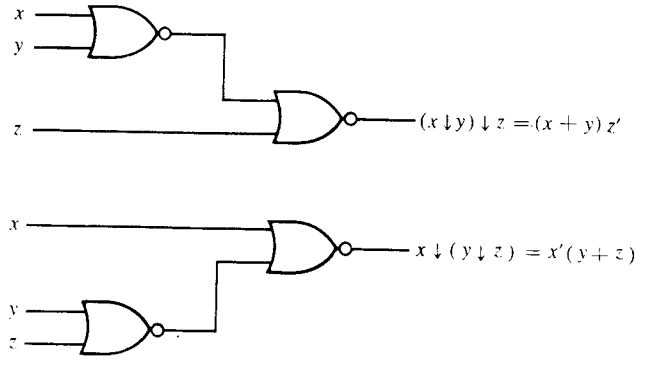
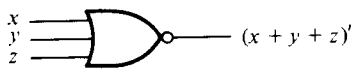
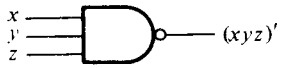


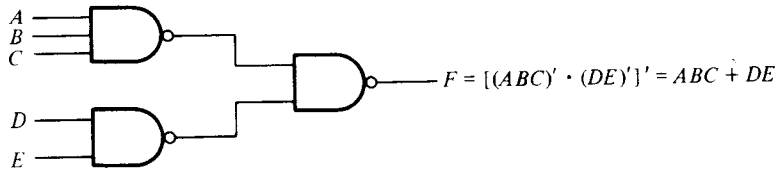
Figura 2-6 Demostración de la no asociatividad del operador NO-O; $(x \downarrow y) \downarrow z \neq x(y \downarrow z)$



(a) Compuerta NOR de tres entradas



(b) Compuerta NAND de tres entradas



(c) Compuertas NAND en cascada

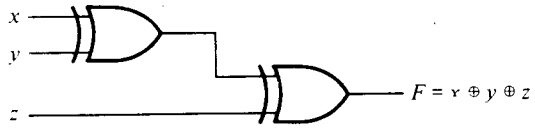
Figura 2-7 Compuertas NOR en cascada y de multi-entrada y compuertas NAND

cuito de la Figura 2-7(c). La función de Boole para el circuito debe escribirse así:

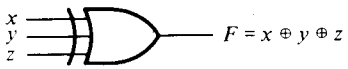
$$F = [(ABC)'(DE)']' = ABC + DE$$

La segunda expresión se obtiene del teorema de De Morgan. Esta muestra que se puede realizar una expresión en suma de productos por medio de compuertas NAND. Posteriormente se tratará sobre las compuertas NAND y NOR en las Secciones 3-6, 4-7 y 4-8.

Las compuertas OR-exclusiva y de equivalencia son ambas conmutativas y asociativas y pueden extenderse a más de dos entradas. Sin embargo las compuertas OR-exclusiva de multientrada no son comunes desde el punto de vista de los circuitos. En efecto, aun una función de dos entradas se construye usualmente con otro tipo de compuertas. Así, la definición de estas funciones debe modificarse cuando se expande a más de dos variables. La función OR-exclusiva es *impar*, es decir, es igual a 1 si las variables de entrada tienen un número impar de unos. La función de equivalencia en una función *par*, es decir, es igual a 1 si las variables de entrada tienen un número par de ceros. La construcción de



(a) Usando compuertas de dos entradas



(b) Una compuerta de tres entradas

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(c) Tabla de verdad

Figura 2-8 Compuerta OR-exclusiva de tres entradas

una función OR-exclusiva de tres entradas se muestra en la Figura 2-8. Esto último se realiza normalmente conectando en cascada compuertas de dos entradas como se muestra en (a). Gráficamente, se puede representar con una sola compuerta de tres entradas como se ilustra en (b). La tabla de verdad en (c) claramente indica que la salida F es igual a 1 si solamente una entrada es igual a 1 o si todas las entradas son igual a 1, es decir, cuando el número total de unos de las variables de entrada es *impar*. Una ulterior discusión sobre el OR-exclusiva y la equivalencia se verán en la Sección 4-9.

2-8 FAMILIAS DE CIRCUITOS INTEGRADOS LOGICO DIGITALES

El circuito integrado se introdujo en la Sección 1-9, donde se dijo que los circuitos digitales se construían invariablemente en circuitos integrados. Después de haber tratado varias compuertas lógicas digitales en la sección anterior, se está en posición de presentar las compuertas de circuitos integrados y de discutir sus propiedades generales.

Las compuertas digitales de circuitos integrados se clasifican no solamente por su operación lógica, sino por la familia de circuitos lógicos, específicos a la cual pertenecen. Cada familia tiene un circuito electrónico básico propio, mediante el cual se desarrollan funciones y circuitos digitales más complejos. El circuito básico en cada familia es o una compuerta NAND ó una NOR. Las compuertas electrónicas usadas en la construcción de circuitos básicos se usan para determinar el nombre de la familia lógica. Hay muchas familias lógicas de circuitos integrados digitales que han sido introducidos comercialmente. Aquellas que han alcanzado buena popularidad se listan a continuación.

TTL	Lógica de transistores (transistor-transistor logic)
ECL	Lógica de acoplamiento de emisor (emitter-coupled logic)
MOS	Semiconductor de óxido de metal (metal-oxide semiconductor)
CMOS	Semiconductor de óxido de metal complementario (complementary metal-oxide semiconductor)
I^2L	Lógica de inyección integrada (integrated-injection logic)

La TTL tiene una lista extensa de funciones digitales y es comúnmente la familia lógica más popular. La ECL se usa en sistemas que requieren operaciones de alta velocidad. Los MOS e I^2L se usan en circuitos que requieren alta densidad de componentes y la CMOS se usa para sistemas que requieren bajo consumo de poder.

El análisis de los circuitos electrónicos básicos en cada familia lógica se representa en el Capítulo 13. El lector que está familiarizado con electrónica básica puede referirse al Capítulo 13 en este punto, con el fin de familiarizarse con estos circuitos electrónicos. Aquí se limitará la discu-

sión a las propiedades generales de las diferentes compuertas en circuitos integrados disponibles comercialmente.

Debido a la alta densidad con la que puedan ser fabricados los transistores con MOS e I^2L, estas dos familias se usan principalmente para funciones LSI. Las otras tres familias TTL, ECL y CMOS se usan en las compuertas LSI y en un gran número de compuertas MSI y SSI. Las compuertas SSI son aquellas que contienen un número pequeño de compuertas o flip-flops (presentadas en la Sección 6-2) en una pastilla de circuito integrado. El límite del número de circuitos en un componente SSI es el número de patillas de la pastilla. Una pastilla de 14 patillas, por ejemplo, puede alojar solamente cuatro compuertas de dos entradas cada una, ya que cada compuerta necesita 3 patillas externas: dos para entradas y una para la salida, para dar un total de 12 patillas. Las dos patillas restantes se usan para el suministro de potencia a los circuitos.

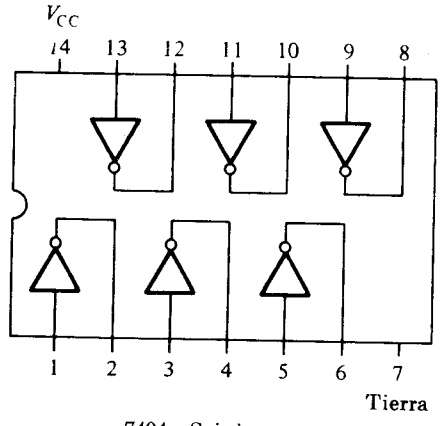
Algunos circuitos SSI se muestran en la Figura 2-9. Cada circuito está encapsulado en una pastilla de 14 o 16 patillas. Las patillas se numeran a lo largo de los dos lados de la pastilla y se especifican las conexiones que pueden hacerse. Las compuertas dibujadas dentro del circuito integrado son para información solamente y no pueden verse ya que en la realidad el circuito integrado aparece de la forma ilustrada en la Figura 1-8.

Los circuitos integrados TTL se distinguen comúnmente por la designación numérica tal como la serie 5400 y 7400. La designación numérica de la serie 7400 implica que los circuitos integrados están numerados 7400, 7401, 7402 etc. Algunos fabricantes tienen circuitos integrados TTL disponibles con diferentes designaciones numéricas tales como la serie 9000 y 8000.

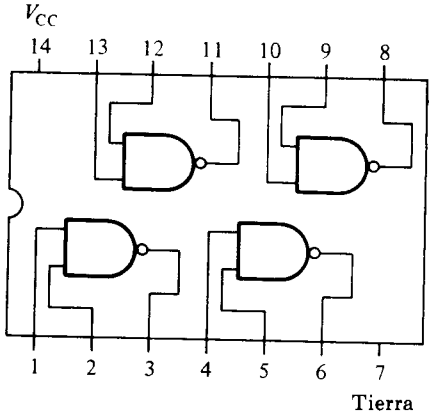
La Figura 2-9(a) ilustra dos circuitos TTL SSI. El 7404 viene con cuatro compuertas NAND de 2 entradas. Los terminales marcados V_{CC} y GND son para las patillas de la fuente del poder que requieren un voltaje de 5 voltios para la adecuada operación.

El tipo ECL más común se designa como la serie 10.000. La Figura 2-9(b) muestra dos circuitos ECL. El 10102 viene con cuatro compuertas NOR de 2 entradas. Nótese que la compuerta ECL puede tener dos entradas, una para la función NOR y la otra para la función OR, (pin 9 del circuito integrado 10102). El circuito integrado 10107 contiene tres compuertas OR-exclusiva, en este caso hay dos salidas de cada compuerta. La otra da la función de NOR-exclusiva o equivalencia. Las compuertas ECL tienen tres terminales para suministro de poder. V_{CC1} y V_{CC2} se conectan por lo general a tierra y V_{EE} a un voltaje de $-5{,}2$ voltios.

Los circuitos CMOS de la serie 4000 se muestran en la Figura 2-9(c). Solamente se pueden acomodar dos compuertas NOR de cuatro entradas en el 4002 debido a la limitación de patillas. El 4050 contiene seis circuitos separadores (buffer). Ambos circuitos integrados tienen dos terminales sin uso, marcados NC (no conexión). El terminal marcado V_{DD} requiere un voltaje de suministro de 3 a 15 voltios y V_{SS} comúnmente se conecta a tierra.

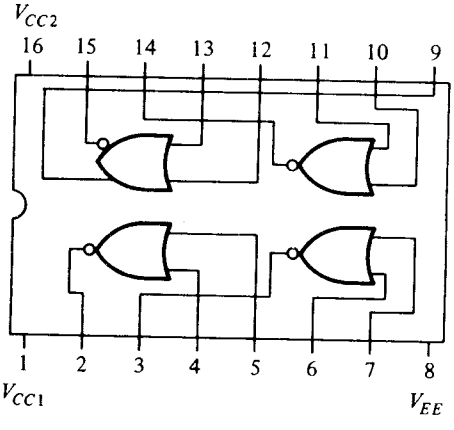


7404—Seis inversores

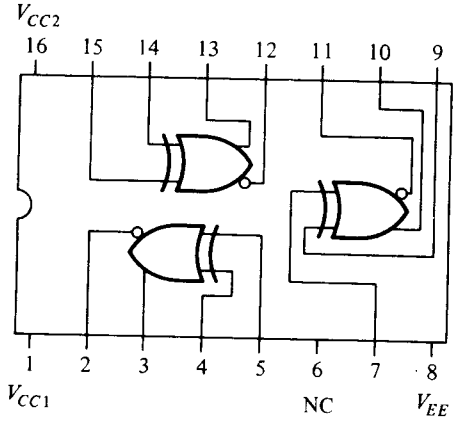


7400—Cuatro compuertas NAND de dos entradas

(a) Compuertas TTL

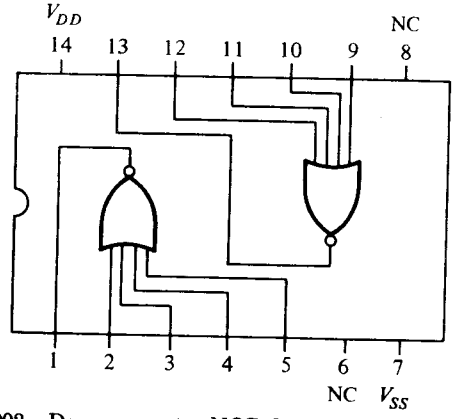


10102—Cuatro compuertas NOR de dos entradas

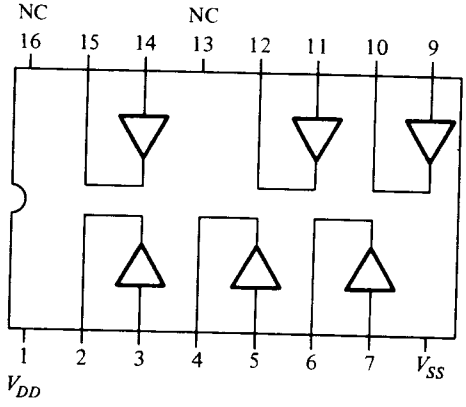


10107—Tres compuertas OR-exclusiva/NOR

(b) Compuertas ECL



4002—Dos compuertas NOR de 4 entradas



4050—Seis separadores

(c) Compuertas CMOS

Figura 2-9 Algunas compuertas típicas en circuitos integrados

Lógica positiva y negativa

La señal binaria a la entrada ó salida de cada compuerta puede tener uno de dos valores, excepto durante la transición. El valor de una señal representa lógica 1 y el otro lógica 0. Como se asignan dos valores de señal a los dos valores lógicos existen dos tipos de señales asignadas a la lógica. Debido al principio de la dualidad en el álgebra de Boole, un intercambio de la asignación de un valor de señal resultará en una función dual.

Considérese los dos valores de la señal binaria mostrada en la Figura 2-10. Un valor debe ser mayor que el otro ya que tienen que ser diferentes para poder distinguirlos. Desígnese el nivel alto como *H* (High) y el nivel bajo como *L* (Low). Hay dos alternativas para la asignación de la lógica.

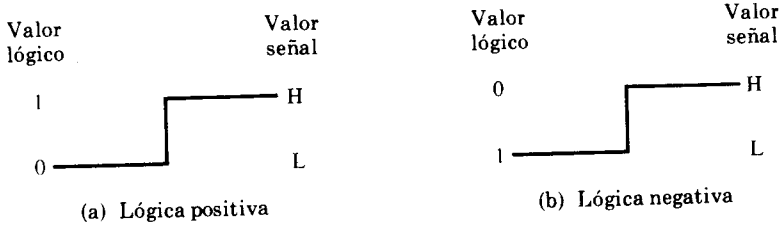


Figura 2-10 Asignación de amplitud de señal y tipo de lógica

Escoger el nivel alto *H* para representar la lógica 1 como se muestra en la Figura 2-10(a) y mediante el cual se define el sistema de lógica *positiva*; O escoger el nivel *L* para representar lógica 1 de la manera ilustrada en la Figura 2-10(b) por medio del cual se define el sistema de *lógica negativa*. Los términos *positivos* y *negativos* no son adecuados ya que ambas señales pueden ser positivas o negativas. No es la polaridad de las señales lo que determina el tipo de lógica sino la asignación de los valores lógicos de acuerdo a las amplitudes relativas de las señales.

Las hojas técnicas de especificación de datos de los circuitos integrados definen funciones digitales no en términos de lógica 1 o lógica 0 sino en términos de niveles *H* y *L*. Se le deja al usuario la oportunidad de usar las asignaciones positiva y negativa. En la Tabla 2-7 se listan los voltajes de nivel alto (*H*) y nivel bajo (*L*) para tres familias de circuitos integrados

Tabla 2-7 Niveles *H* y *L* en las familias de CI lógicos

Tipo de familia de CI	Voltaje de fuente (V)	Nivel alto de voltaje (V)		Nivel bajo de voltaje (V)	
		Rango	Típico	Rango	Típico
TTL	V_{CC} = 5	2,4 - 5	3,5	0 - 0,4	0,2
ECL	V_{EE} = −5,2	−0,95 - −0,7	−0,8	−1,9 - −1,6	−1,8
CMOS	V_{DD} = 3 - 10	V_{DD}	V_{DD}	0 - 0,5	0
Lógica positiva:			lógica 1		lógica 0
Lógica negativa:			lógica 0		lógica 1

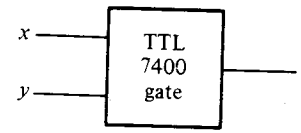
lógicos digitales. En cada familia hay un rango de valores de voltaje que el circuito puede reconocer como nivel alto o nivel bajo. El valor típico es el que se usa más comúnmente. La tabla da además los voltajes de suministro como referencia para cada familia.

TTL tiene valores típicos de $H = 3{,}5$ voltios y $L = 0{,}2$ voltios. ECL tiene dos valores negativos en $H = 0{,}8$ a $L = -1{,}8$ voltios. Nótese que a pesar de ser de dos voltajes negativos el más alto es $-0{,}8$. Las compuertas CMOS pueden usar un voltaje de suministro V_{DD} en el rango de 3 a 15 voltios con voltajes típicos de 5 a 10 voltios. Los valores de la señal en CMOS son función del voltaje de suministro con $H = V_{DD}$ y $L = 0$ voltios. La asignación de las polaridades para la lógica positiva y negativa se indican también en la tabla.

Después del anterior planteamiento, se hace necesario justificar los símbolos lógicos usados por los circuitos integrados mostrados en la Figura 2-9. Tómese por ejemplo, una de las compuertas del circuito integrado 7400. El diagrama de bloque de la compuerta se muestra en la Figura 2-11(b). La tabla de verdad del fabricante de la compuerta dada en la hoja de especificaciones se muestra en la Figura 2-11(a). Esto especifica el

x	y	z
L	L	H
L	H	H
H	L	H
H	H	L

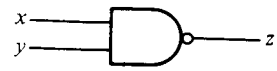
(a) Tabla de verdad en términos de H y L.



(b) Diagrama de bloque de la compuerta

x	y	z
0	0	1
0	1	1
1	0	1
1	1	0

(c) Tabla de verdad para la lógica positiva: $H = 1, L = 0$.



(d) Símbolo gráfico para la compuerta NAND de lógica positiva.

x	y	z
1	1	0
1	0	0
0	1	0
0	0	1

(e) Tabla de verdad de lógica negativa: $L = 1, H = 0$.

(f) Símbolo gráfico para la compuerta NOR de lógica negativa.

Figura 2-11 Demostración de lógica positiva y lógica negativa

comportamiento físico de la compuerta con H con un valor típico de 3,5 voltios y L de 0,2 voltios. Esta compuerta física puede funcionar como una compuerta NAND ó como una compuerta NOR dependiendo de la asignación de la polaridad.

La tabla de verdad de la Figura 2-11(c) asume lógica positiva con $H = 1$ y $L = 0$. Al comparar la tabla de verdad con las tablas de verdad de la Figura 2-5, se reconoce que se trata de una compuerta NAND. El símbolo gráfico para una compuerta NAND de lógica positiva se muestra en la Figura 2-11(d) y es similar a la que se ha adaptado previamente.

Ahora, considérese una asignación de lógica positiva a esta compuerta física con $L = 1$ y $H = 0$. El resultado es la tabla de verdad mostrada en la Figura 2-11(e). Esta tabla se reconoce que representa la función NOR a pesar de que sus entradas estén listadas al revés. El símbolo gráfico para una compuerta NOR de lógica negativa se muestra en la Figura 2-11(f). El pequeño triángulo en los alambres de entrada y salida designan un *indicador de polaridad*. La presencia de este indicador de polaridad en las entradas y salidas indican que la lógica negativa se asigna al terminal. Así, la misma compuerta física puede funcionar o como NAND de lógica positiva o como NOR de lógica negativa. El uno dibujado en el diagrama es completamente dependiente de la asignación de polaridad que el diseñador desea emplear.

De manera similar, es posible demostrar que la NOR de lógica positiva es la misma compuerta física que la NAND de lógica negativa. La misma relación es válida entre las compuertas AND y OR o entre las compuertas OR-exclusiva y la de equivalencia. En cualquier caso si se asume lógica negativa en cualquier terminal de entrada o salida es necesario incluir el triángulo indicador de polaridad a lo largo del terminal. Algunos diseñadores digitales usan esta convención para facilitar el diseño de los circuitos digitales cuando se usan exclusivamente las compuertas NOR y NAND. En este libro no se usará esta simbología pero se recurrirá a otros métodos para hacer diseños con compuertas NAND y NOR. Nótese que los CI presentados en la Figura 2-9 se muestran con sus símbolos gráficos de lógica positiva. Se hubieran podido mostrar con sus símbolos lógicos negativos si se hubiera deseado.

La conversión de lógica positiva a lógica negativa y viceversa, es esencialmente una operación que cambia unos a ceros y ceros a unos en las entradas y salidas de la compuerta. Debido a que esta operación produce una función dual, el cambio de todos los terminales de una polaridad a otra dará el mismo resultado que tomar el dual de la función. El resultado de esta conversión es que todas las operaciones AND se convierten a operaciones OR y viceversa. Además, no se debe olvidar el incluir el indicador de polaridad en los símbolos gráficos cuando se asume lógica negativa.

El pequeño triángulo que representa un indicador de polaridad y el pequeño círculo que representa una complementación tienen efectos similares, pero significados diferentes, por tanto, pueden remplazarse el uno por el otro, si se tiene en cuenta que su interpretación es diferente. Un

círculo seguido por un triángulo, tal como en la Figura 2-11(f), representa una complementación seguida de un indicador de polaridad de lógica negativa. Los dos se cancelan entre sí y pueden quitarse. Pero si se quitan ambos, las entradas y salidas de la compuerta representarán polaridades diferentes.

Características especiales

Las características de las familias de CI lógico digitales se comparan analizando el circuito de la compuerta básica de cada familia. Los parámetros más importantes que son evaluados y comparados son fan-out, disipación de poder, demora de propagación y margen de ruido. Se explicará primero las propiedades de estos parámetros para luego usarlos para comparar las familias lógicas de CI.

Fan-out especifica el número de cargos normales que puede accionar la salida de la compuerta sin menoscabar su operación normal. Una carga normal se define como la cantidad de corriente necesitada para la entrada de otra compuerta en la misma familia de CI. Algunas veces se usa el término *cargado* en vez de fan-out. Este término se deduce del hecho de que la salida de la compuerta suministra una cantidad limitada de corriente por encima de la cual no opera correctamente y se dice por este caso que está sobrecargada. La salida de la compuerta generalmente se conecta a las entradas de otras compuertas similares. Cada entrada consume cierta cantidad de potencia de la compuerta de entrada de tal manera que cada conexión adicional se agrega a la carga de la compuerta. "Las reglas de carga" se listan comúnmente para una familia de circuitos digitales normalizados. Estas reglas especifican la máxima cantidad de carga permisible para cada salida de cada circuito. Al excederse la carga máxima especificada se podría causar mal funcionamiento ya que el circuito no puede suministrar el poder demandado. El fan-out es el número máximo de entradas que pueden conectarse a la salida de la compuerta y se expresa con un número.

Las capacidades de fan-out de la compuerta deben considerarse cuando se simplifican las funciones de Boole. Se debe tener mucho cuidado de no desarrollar expresiones que resulten en una compuerta con sobrecarga. Los amplificadores no inversores o separados se usan para suministrar capacidad adicional de accionamiento para el caso de cargas pesadas.

Disipación de potencia es la potencia suministrada necesaria para operar la compuerta. Este parámetro se expresa en milivatios (mW) y representa la potencia real designada por la compuerta. El número que representa este parámetro no incluye la potencia suministrada de otra compuerta o sea que representa la potencia suministrada a la compuerta por la fuente de poder. Un CI con cuatro compuertas exigirá de la fuente cuatro veces la potencia disipada por cada compuerta. En un sistema dado puede haber muchos circuitos integrados y sus potencias deben tenerse en cuenta. El poder total disipado en un sistema es la suma total del poder disipado de todos los CI.

Retardo de propagación es el tiempo promedio de demora en la transición de programación de una señal de la entrada a la salida, cuando las

señales binarias cambian de valor. Las señales en una compuerta toman cierta cantidad de tiempo para propagarse de las entradas a la salida. Este intervalo de tiempo se define como la demora de propagación de la compuerta. Esta última se expresa en nanoseconds (ns). Un ns es igual a 10^{-9} segundos.

Las señales que viajan de las entradas de un circuito digital a las salidas pasan por una serie de compuertas. La suma de las demoras de propagación a través de las compuertas es la demora total de propagación del circuito. Cuando la velocidad de operación es importante, cada compuerta debe tener una pequeña demora de propagación y el circuito digital debe tener un número mínimo de compuertas en serie entre las entradas y las salidas.

Las entradas digitales en la mayoría de los circuitos digitales se aplican simultáneamente a más de una compuerta. Todas aquellas compuertas que reciben sus entradas exclusivamente de entradas externas constituyen el primer nivel de lógica del circuito. Las compuertas que reciben al menos una entrada, a partir de una salida de una compuerta del primer nivel de lógica, se consideran en el segundo nivel de lógica y de manera similar para los niveles tercero y superiores. La demora total de propagación del circuito es igual a la demora de propagación de la compuerta por el número de niveles lógicos en el circuito. Así, una reducción en el número de niveles lógicos dará como resultado una reducción de la demora de la señal y circuitos más rápidos. La reducción de la demora de propagación en los circuitos podría ser más importante que la reducción en el número total de compuertas en el caso de que la velocidad de operación sea el factor preponderante.

Margen de ruido es el máximo voltaje de ruido agregado a la señal de entrada de un circuito digital que no cause un cambio indeseable, a la salida del circuito. Hay dos tipos de ruido que deben considerarse. El ruido (DC) CD causado por la desviación en los niveles de voltaje de señal. El ruido CA (AC) es el pulso aleatorio que puede ver creado por otras señales conmutadas. Así, el ruido es el término usado para denotar una señal indeseable superimpuesta a una señal de operación normal. La habilidad de los circuitos para operar con confiabilidad en un ambiente de ruido es importante en muchas aplicaciones. El margen de ruido se expresa en voltios (V) y representa la máxima señal de ruido que puede ser tolerada por una compuerta.

Características de las familias de CI lógicos

El circuito básico de la familia lógica de TTL es la compuerta NAND. Hay muchas versiones de TTL de las cuales se listan tres en la Tabla 2-8. Esta tabla da las características generales de las familias de CI lógicos. Los valores listados son representativos con base en la comparación. Para cualquier familia o versión los valores pueden variar.

La compuerta TTL normalizada fue la primera versión de la familia TTL. Se agregaron mejoras a medida que la tecnología ha progresado. La TTL Schottky es una de las últimas innovaciones que reducen la demora de propagación pero que resulta en un aumento de asignación de

Tabla 2-8 Características de familias de CI lógicos

Familia de CI lógico	Fan-out	Disipación de potencia en (mW)	Demora de propagación (ns)	Margen de ruido (V)
TTL normalizada	10	10	10	0,4
TTL Schottky	10	22	3	0,4
TTL Schottky de baja potencia	20	2	10	0,4
ECL	25	25	2	0,2
CMOS	50	0,1	25	3

potencia. La versión TTL Schottky de baja potencia sacrifica alguna velocidad para reducir la disipación de potencia. Esta última tiene la misma demora de propagación que la TTL normalizada pero con una disipación de potencia bastante reducida. El fan-out de la versión TTL normalizada es 10 pero la versión Schottky de baja potencia tiene un fan-out de 20. Bajo ciertas condiciones las otras versiones podrían tener un fan-out de 20. El margen de ruido es mejor que 0,4 V, con un valor típico de 1V.

El circuito básico de la familia ECL es la compuerta NOR. La ventaja especial de la compuerta ECL es la poca demora de propagación. Algunas versiones de ECL pueden tener una demora de propagación corta del orden de 0,5 ns. La disipación de potencia en las compuertas ECL es comparativamente alta y su margen de ruido bajo. Estos últimos dos parámeros son una desventaja al escoger la familia ECL con respecto a las demás. Pero, a pesar de su baja demora de propagación la ECL ofrece la más alta velocidad de todas las familias y es un último recurso para sistemas rápidos.

El circuito básico de CMOS es l inversor con el cual se pueden construir las compuertas NAND y NOR. La ventaja especial del CMOS es su designación de potencia extremadamente baja. Bajo condiciones estáticas la disipación de potencia de una compuerta CMOS es despreciable ya que promedia 10 nV (nW). Cuando la señal de la compuerta cambia de estado, hay una disipación de potencia dinámica proporcional a la frecuencia al cual el circuito está expuesto. El número listado en la tabla es un valor típico de la disipación de potencia dinámica en las compuertas CMOS.

La mayor desventaja de las CMOS es su alta demora de propagación. Esto significa que no es práctico usarlas en sistemas que requieren operaciones de alta velocidad. Los parámetros característicos para la compuerta CMOS dependen del voltaje V_{DD} de la fuente de poder que se use. La disipación de potencia crece con el aumento del voltaje de suministro. La demora de propagación disminuye con el aumento del voltaje de suministro y el margen del ruido se estima en un 40% del valor del primero.

REFERENCIAS

1. Boole, G., *An Investigation of the Laws of Thought*. Nueva York: Dover Pub., 1954.

2. Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits". *Trans. of the AIEE*, Vol. 57 (1938), 713-23.

3. Huntington, E. V., "Sets of Independent Postulates for the Algebra of Logic". *Trans. Am. Math. Soc.*, Vol. 5 (1904), 288-309.

4. Birkhoff, G., y T. C. Bartee, *Modern Applied Algebra*. Nueva York: McGraw-Hill Book Co., 1970.

5. Birkhoff, G., y S. Maclane, *A Survey of Modern Algebra*, 3a. ed. Nueva York: The Macmillan Co., 1965.

6. Hohn, F. E., *Applied Boolean Algebra*, 2a. ed. Nueva York: The Macmillan Co., 1966.

7. Whitesitt, J. E., *Boolean Algebra and its Applications*. Reading, Mass.: Addison-Wesley Pub. Co., 1961.

8. *The TTL Data Book for Design Engineers*. Dallas, Texas: Texas Instruments Inc., 1976.

9. *MECL Integrated Circuits Data Book*. Phoenix, Ariz.: Motorola Semiconductor Products, Inc., 1972.

10. *RCA Solid State Data Book Series: COS/MOS Digital Integrated Circuits*. Somerville, N. J.: RCA Solid State Div., 1974.

PROBLEMAS

2-1. ¿Cuál de las seis leyes básicas (conjunto cerrado asociativa, conmutativa, de identidad, inversa y distributiva) son cumplidas por el par de operadores binarios listados a continuación?

+	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

·	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

2-2. Demuestre que el conjunto de los tres elementos $\{0, 1, 2\}$ y los dos operadores binarios $+$ y $\cdot$ de la manera definida en la tabla anterior, no constituyen el álgebra de Boole. Establezca cuál de los postulados de Huntington no se cumple.

2-3. Demuestre por medio de tablas de verdad la validez de los siguientes teoremas del álgebra de Boole.

(a) Las leyes asociativas.

(b) Los teoremas de De Morgan para tres variables.

(c) La ley distributiva de $+$ sobre $\cdot$.

2-4. Repita el Problema 2-3 usando los diagramas de Venn.

2-5. Simplifique las siguientes funciones de Boole al menor número de literales.

(a) $xy + xy'$

(b) $(x + y)(x + y')$

(c) $xyz + x'y + xyz'$

(d) $zx + zx'y$

(e) $(A + B)'(A' + B')'$

(f) $y(wz' + wz) + xy$

2-6. Reduzca las siguientes expresiones de Boole al número de literales solicitado al frente de cada una de ellas.

(a) $ABC + A'B'C + A'BC + ABC' + A'B'C'$ — a cinco literales

(b) $BC + AC' + AB + BCD$ — a cuatro literales

(c) $[(CD)' + A]' + A + CD + AB$ — a tres literales

(d) $(A + C + D)(A + C + D')(A + C' + D)(A + B')$ — a cuatro literales

2-7. Encuentre el complemento de las siguientes funciones de Boole y redúzcalas al mínimo número de literales.

(a) $(BC' + A'D)(AB' + CD')$

(b) $B'D + A'BC' + ACD + A'BC$

(c) $[(AB)'A][(AB)'B]$

(d) $AB' + C'D'$

2-8. Dadas dos funciones de Boole F_1 y F_2:

(a) Demuestre que la función de Boole $E = F_1 + F_2$, obtenida al aplicar la función OR a las dos funciones, contiene la suma de todos los términos mínimos en F_1 y F_2.

(b) Demuestre que la función de Boole $G = F_1F_2$, obtenida al aplicar la función AND a las dos funciones, contiene los términos mínimos comunes a ambas F_1 y F_2.

2-9. Obtenga la tabla de verdad de la siguiente función:

$$F = xy + xy' + y'z$$

2-10. Exprese las funciones de Boole simplificadas del Problema 2-6 con compuertas lógicas.

2-11. Dada la función de Boole:

$$F = xy + x'y' + y'z$$

(a) Exprésela con compuertas AND, OR y NOT.

(b) Exprésela con compuertas OR y NOT *solamente*.

(c) Exprésela con compuertas AND y NOT *solamente*.

2-12. Simplifique las funciones T_1 y T_2 al mínimo número de literales.

A	B	C	T_1	T_2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	0	1

2-13. Exprese las siguientes funciones en suma de términos mínimos y producto de términos máximos.

(a) $F(A, B, C, D) = D(A' + B) + B'D$

(b) $F(w, x, y, z) = y'z + wxy' + wxz' + w'x'z$

(c) $F(A, B, C, D) = (A + B' + C)(A + B')(A + C' + D')$
$(A' + B + C + D')(B + C' + D')$

(d) $F(A, B, C) = (A' + B)(B' + C)$

(e) $F(x, y, z) = 1$

(f) $F(x, y, z) = (xy + z)(y + xz)$

2-14. Convierta las siguientes expresiones a la otra forma:

(a) $F(x, y, z) = \Sigma(1, 3, 7)$

(b) $F(A, B, C, D) = \Sigma(0, 2, 6, 11, 13, 14)$

(c) $F(x, y, z) = \Pi(0, 3, 6, 7)$

(d) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 6, 12)$

2-15. ¿Cuál es la diferencia entre la forma canónica y la forma normalizada? ¿Cuál es la forma preferible, cuando se expresa una función de Boole con compuertas? ¿Cuál es la forma que se obtiene al leer una función de una tabla de verdad?

2-16. La suma de todos los términos mínimos de una función de Boole de n variables es 1.

(a) Pruebe la anterior afirmación para $n = 3$.

(b) Sugiera un procedimiento para una prueba general.

2-17. El producto de todos los términos máximos de una función de Boole de n variables es 0.

(a) Pruebe la anterior afirmación para $n = 3$.

(b) Sugiera un procedimiento para una prueba general. ¿Se puede usar el principio de dualidad después de probar la parte (b) del Problema 2-16?

2-18. Demuestre que el dual de la OR-exclusiva es igual a su complemento.

2-19. Por sustitución de la función de Boole equivalente a las funciones binarias definidas en la Tabla 2-6 demuestre que:

(a) Los operadores de inhibición e implicación no son ni conmutativos ni asociativos.

(b) Los operadores OR-exclusiva y de equivalencia son conmutativos y asociativos.

(c) El operador NAND no es asociativo.

(d) Los operadores NOR y NAND no son distributivos.

2-20. Una compuerta *mayorista* es un circuito digital cuya salida es 1 si la mayoría de las entradas son 1. De otra forma la salida será 0. Por medio de una tabla de verdad. Encuentre una función de Boole llevada a cabo con una compuerta mayoritaria de 3 entradas. Simplifique la función.

2-21. Verifique la tabla de verdad de la compuerta OR-exclusiva de 3 entradas listada en la Figura 2-8(c). Haga la lista de las ocho combinaciones de x, y y z. Evalúe $A = x \oplus y$ y luego $F = A \oplus z = x \oplus y \oplus z$.

2-22. El SSI de TTL viene mayormente en pastillas de 14 patillas. Se reservan 2 patillas para la fuente de poder y el resto para los terminales de entrada y salida. ¿Cuántas compuertas están encapsuladas en una pastilla de este estilo si contiene el siguiente tipo de compuertas?

(a) Compuertas OR-exclusivas de 2 entradas.

(b) Compuertas AND de 3 entradas.

(c) Compuertas NAND de 4 entradas.
(d) Compuertas NOR de 5 entradas.
(e) Compuertas NAND de 8 entradas.

2-23. Demuestre que una compuerta AND de lógica positiva es una compuerta OR de lógica negativa y viceversa.

2-24. Una familia de CI lógicos tiene compuertas NAND con un fan-out de 5 y compuertas separadas con un fan-out de 10. Demuestre cómo una señal de salida de una compuerta NAND sencilla puede ser aplicada a 50 entradas de compuertas.

Simplificación de funciones de Boole

3

3-1 EL METODO DEL MAPA

La complejidad de las compuertas lógicas digitales con que se llevan a cabo las funciones de Boole se relacionan directamente con la complejidad de la expresión algebraica de la cual se desprende la función. Aunque la representación de la tabla de verdad de una función única, puede aparecer de muchas formas diferentes. Las funciones de Boole pueden ser simplificadas por medios algebraicos de la manera vista en la Sección 2-4. Sin embargo el procedimiento de minimización es un tanto raro ya que carece de reglas específicas para predecir cada paso sucesivo en el proceso de manipulación. El método del mapa presenta un procedimiento simple y directo para minimizar las funciones de Boole. Este método puede ser tratado no solamente en la forma pictórica de una tabla de verdad, sino como una extensión del diagrama de Venn. El método del mapa, propuesto primero por Veitch (1) y modificado ligeramente por Karnaugh (2), se conoce como el "diagrama de Veitch" o el "mapa de Karnaugh".

El mapa es un diagrama, hecho de cuadros. Cada cuadro representa un término mínimo. Como cualquier función de Boole puede ser expresada como una suma de términos mínimos, se desprende que dicha función, se reconoce gráficamente en el mapa a partir del área encerrada por aquellos cuadros cuyos términos mínimos se incluyen en la función. De hecho, el mapa presenta un diagrama visual de todas las formas posibles en que puede ser expresada una función en la forma normalizada. Al reconocer varios patrones, el usuario puede derivar expresiones algebraicas alternas para la misma función de las cuales se puede escoger la más simple. Se asume que la expresión algebraica más simple es cualquiera en una suma de productos o producto de sumas que tiene el mínimo número de literales. (Esta expresión no es necesariamente única.)

3-2 MAPAS DE DOS Y TRES VARIABLES

Un mapa de dos variables se muestra en la Figura 3-1. En él hay cuatro términos mínimos para dos variables, es decir que el mapa consiste en

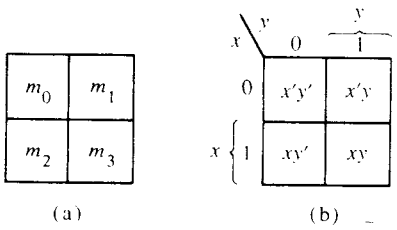


Figura 3-1 Mapa de dos variables

cuatro cuadrados, uno para cada término mínimo. El mapa que se dibuja de nuevo en (b) sirve para demostrar la relación entre los cuadrados y las dos variables. Los ceros y unos marcados para cada fila y columna designan los valores x y y respectivamente. Nótese que la x aparece tildada en la fila 0 y no tildada en la fila 1. De manera similar, y aparece tildada en la columna 0 y no tildada en la columna 1.

Si se marcan los cuadrados cuyos términos mínimos pertenecen a una función dada, el mapa de dos variables se convierte en otro método útil para representar una cualquiera de las 16 funciones de Boole de dos variables. Como ejemplo, la función xy se muestra en la Figura 3-2(a). Como xy es igual a m_3, se coloca un 1 dentro del cuadrado que pertenece a m_3. De manera similar, la función $x + y$ se representa en el mapa de la Figura 3-2(b) por medio de tres cuadrados marcados con unos. Estos cuadrados se escogen de los términos mínimos de la función:

$$x + y = x'y + xy' + xy = m_1 + m_2 + m_3$$

Los tres cuadrados pudieron haberse determinado de la intersección de la variable x en la segunda fila y la variable y en la segunda columna, lo cual cubre el área perteneciente a x o y.

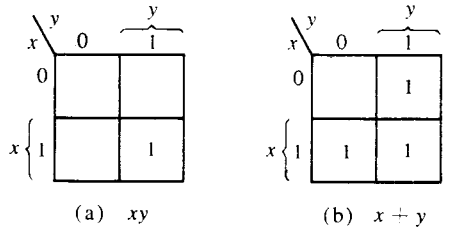


Figura 3-2 Representación de las funciones en un mapa

En la Figura 3-3 se ilustra un mapa de tres variables. Hay ocho términos mínimos para las tres variables. El mapa por tanto consiste en ocho cuadrados. Nótese que los términos mínimos se arreglan, no en secuencia binaria sino en una secuencia similar al código reflejado listado en la Tabla 1-4. La característica de esta secuencia es que solamente un bit cambia de 1 a 0 o de 0 a 1, en la secuencia del listado. El mapa dibujado en la parte (b) se marca con los números de cada fila o cada columna para mostrar la relación entre los cuadrados de las tres variables. Por ejemplo, el cuadrado asignado a m_5 corresponde a la fila 1 y columna 01. Cuando se con-

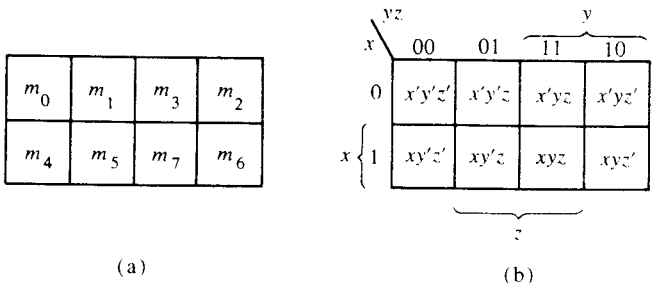


Figura 3-3 Mapa de tres variables

catenan estos dos números darán el número binario 101, cuyo equivalente decimal es 5. Otra manera de mirar el cuadrado $m_5 = xy'z$ es considerar que está en la fila marcada x y en la columna que pertenece a $y'z$ (columna 01). Nótese que hay cuatro cuadrados donde cada variable es igual a 1 y cuatro donde cada una es igual a 0. La variable aparece no tildada en aquellos cuatro cuadrados donde sea igual a 1 y tildada en aquellos que sea igual a 0. Por conveniencia, se escribe la variable usando un símbolo de letra que abarca aquellos cuatro cuadrados donde la primera no esté tildada.

Para entender la utilidad del mapa en la simplificación de funciones de Boole, se debe reconocer la propiedad básica que tienen los cuadrados adyacentes. Cualquier par de cuadrados adyacentes en el mapa difieren por una variable tildada en un cuadrado y no tildada en el otro. Por ejemplo, m_5 y m_7 están en dos cuadrados adyacentes. La variable y está tildada en m_5 y no tildada en m_7, mientras que las otras dos variables son iguales en ambos cuadrados. A partir de los postulados del álgebra de Boole, se desprende que la suma de los términos mínimos en cuadrados adyacentes pueden ser simplificados a un simple término AND consistente en dos literales. Para aclarar lo anterior, considérese la suma de dos cuadrados adyacentes tales como m_5 y m_7:

$$m_5 + m_7 = xy'z + xyz = xz(y' + y) = xz$$

Aquí los dos cuadrados difieren en la variable y, que puede ser removida cuando se forme la suma de los términos mínimos. Así, a cualquier par de términos mínimos en cuadrados adyacentes a los cuales se le aplica la función OR se les causará la remoción de la variable diferente. El siguiente ejemplo explica el procedimiento para minimizar una función de Boole con un mapa.

EJEMPLO 3-1: Simplificar la función de Boole:

$$F = x'yz + x'yz' + xy'z' + xy'z$$

Primero, se marca un 1 en cada cuadrado cuando sea necesario para representar la función de la manera mostrada en la Figura 3-4. Esto puede lograrse de dos maneras: convirtiendo cada término a un número binario para luego marcar 1 en el cuadrado corres-

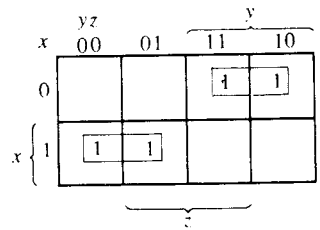


Figura 3-4 Mapa del Ejemplo 3-1; $x'yz + x'yz' + xy'z' + xy'z = x'y + xy'$

pondiente u obteniendo la coincidencia de las variables en cada término. Por ejemplo, el término $x'yz$ tiene su correspondiente número binario 011 y representa el término mínimo m_3 en el cuadrado 011. La segunda forma de reconocer el cuadrado es por coincidencia de las variables x', y y z, las cuales se encuentran en el mapa observando que x' pertenece a los cuatro cuadrados de la primera fila, y pertenece a los cuatro cuadrados de las dos columnas de la derecha y z pertenece a los cuatro cuadrados de las dos columnas del medio. El área que pertenece a los tres literales es el cuadrado de la primera fila y la tercera columna. De igual manera, los otros tres cuadrados que pertenecen a la función F se marcan con un 1 en el mapa. Se representa así la función en el área que contiene cuatro cuadrados, cada uno marcado con un 1, de la manera mostrada en la Figura 3-4. El siguiente paso es subdividir el área dada en cuadrados adyacentes. Estos se indican en el mapa por medio de dos rectángulos, cada uno conteniendo dos unos. El rectángulo superior derecho representa el área encerrada por $x'y$; el inferior izquierdo el área encerrada por xy'. La suma de estos dos términos dará la respuesta:

$$F = x'y + xy'$$

Seguidamente considérese los dos cuadrados marcados m_0 y m_2 en la Figura 3-3(a) o $x'y'z'$ y $x'yz'$ en la Figura 3-3(b). Estos dos términos mínimos también difieren en una variable y y su suma puede ser simplificada a una expresión de dos literales:

$$x'y'z' + x'yz' = x'z'$$

En consecuencia, se puede modificar la definición de los cuadrados adyacentes para incluir este y otros casos similares. Esto se logra considerando el mapa como un dibujo en una superficie donde los bordes izquierdo y derecho se tocan entre sí para formar cuadrados adyacentes.

EJEMPLO 3-2: Simplificar la función de Boole:

$$F = x'yz + xy'z' + xyz + xyz'$$

El mapa de esta función se muestra en la Figura 3-5. Hay cuatro cuadrados marcados con 1, para cada uno de los términos míni-

mos de la función. Dos cuadrados adyacentes se combinan en la tercera columna para dar un término de dos literales yz. Los dos cuadrados restantes con 1, son adyacentes por la nueva definición y se muestran en dos cuadrados que cuando se combinan darán un término de dos literales xz'. La función simplificada será:

$$F = yz + xz'$$

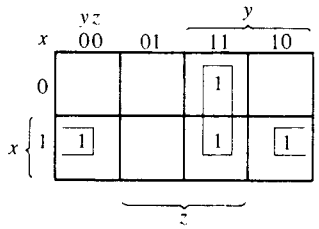


Figura 3-5 Mapa del Ejemplo 3-2; $x'yz + xy'z' + xyz + xyz' = yz + xz'$

Considérese ahora cualquier combinación de cuatro cuadrados adyacentes en el mapa de tres variables. Una combinación como ésta representa la aplicación de la función OR de cuatro términos mínimos adyacentes y que resulta en una expresión de un literal solamente. Por ejemplo, la suma de cuatro términos mínimos adyacentes m_0, m_2, m_4 y m_6, se reduce al solo literal z' como se muestra a continuación:

$$\begin{aligned} x'y'z' + x'yz' + xy'z' + xyz' &= x'z'(y' + y) + xz'(y' + y) \\ &= x'z' + xz' = z'(x' + x) = z' \end{aligned}$$

EJEMPLO 3-3: Simplificar la función de Boole:

$$F = A'C + A'B + AB'C + BC$$

El mapa para simplificar esta función se muestra en la Figura 3-6. Algunos de los términos de la función tienen menos de tres literales y son representados en el mapa por más de un cuadrado. Así, para encontrar los cuadrados correspondientes a $A'C$ se forma la coincidencia de A' (primera fila) y C (dos columnas del medio) y se obtienen los cuadrados 001 y 011. Nótese que al enmarcar los unos con cuadrados es posible encontrar un uno ya

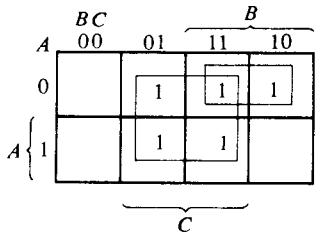


Figura 3-6 Mapa del Ejemplo 3-3; $A'C + A'B + AB'C + BC = C + A'B$

colocado en el término anterior. En este ejemplo, el segundo término $A'B$ tiene unos en los cuadrados 011 y 010, pero el cuadrado 011 es común al primer término $A'C$ y solamente contiene un uno. La función de este ejemplo tiene cinco términos mínimos, como se indica por los cinco cuadrados marcados con un 1. Se simplifica combinando cuatro cuadrados del centro para dar el literal C. El cuadrado restante marcado con 1 en 010 se combina con un cuadrado adyacente que ya ha sido usado una vez. Esto es permisible y aun deseable ya que la combinación de los dos cuadrados da el término $A'B$ mientras que el término mínimo sencillo representado por el cuadrado da el término $A'BC'$ de 3 variables. La función simplificada es:

$$F = C + A'B$$

EJEMPLO 3-4: Simplifíquese la función de Boole:

$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$

Aquí se han dado los términos mínimos por medio de números decimales. Los cuadrados correspondientes se marcan con unos de la manera mostrada en la Figura 3-7. Del mapa se obtiene la función simplificada:

$$F = z' + xy'$$

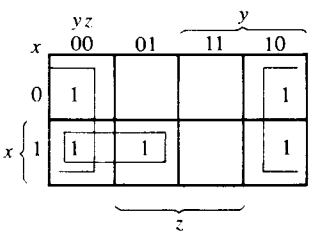


Figura 3-7 $f(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

3-3 MAPA DE CUATRO VARIABLES

El mapa para las funciones de Boole de cuatro variables binarias se muestra en la Figura 3-8. En (a) se listan los 16 términos mínimos y los cuadrados asignados a cada uno. En (b) se redibuja el mapa para demostrar la relación con las cuatro variables. Las columnas y las filas se enumeran en la secuencia del código reflejado con un dígito que cambia de valor entre dos columnas o filas adyacentes. El término mínimo correspondiente a cada cuadrado puede obtenerse por la concatenación del número de la fila con el número de la columna. Así, los números en la tercera fila (11) y la segunda columna (01) una vez concatenados, dan el número binario 1101, equivalente binario al decimal 13. Por tanto, el cuadrado en la tercera fila y la segunda columna representa el término mínimo m_{13}.

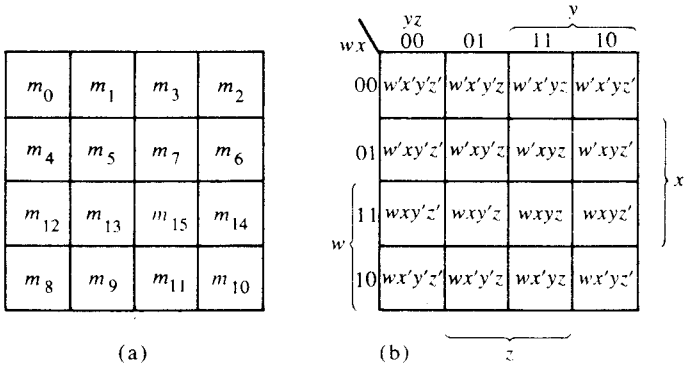


Figura 3-8 Mapa de cuatro variables

La minimización, por medio del mapa, de una función de Boole de cuatro variables, es similar al método usado para minimizar funciones de tres variables. Los cuadrados adyacentes se definen como cuadrados cercanos entre sí. Además, se considera el mapa que yace en una superficie con los bordes superior e inferior y los bordes izquierdo y derecho tocándose entre sí para formar cuadrados adyacentes. Por ejemplo, m_0 y m_2 forman cuadrados adyacentes de la misma forma que m_3 y m_1. La combinación de cuadrados adyacentes, útil durante el proceso de simplificación, se determina fácilmente por inspección del mapa de cuatro variables:

Un cuadrado representa un término mínimo, dando un término de cuatro literales.

Dos cuadrados adyacentes representan un término de tres literales.

Cuatro cuadrados adyacentes representan un término de dos literales.

Ocho cuadrados adyacentes representan un término de un literal.

Dieciséis cuadrados adyacentes representan la función igual a 1.

Ninguna otra combinación de cuadrados pueden simplificar la función. Los siguientes ejemplos muestran el procedimiento usado para simplificar las funciones de Boole de cuatro variables.

EJEMPLO 3-5: Simplifíquese la función de Boole:

$$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$$

Como la función tiene cuatro variables, se debe usar un mapa de cuatro variables. Los términos mínimos listados en la suma se marcan con unos en el mapa de la Figura 3-9. Ocho cuadrados adyacentes marcados con unos pueden combinarse para formar un término literal y'. Los restantes tres unos a la derecha no pueden combinarse entre sí para dar un término simplificado. Deben combinarse como dos o cuatro cuadrados adyacentes. Entre ma-

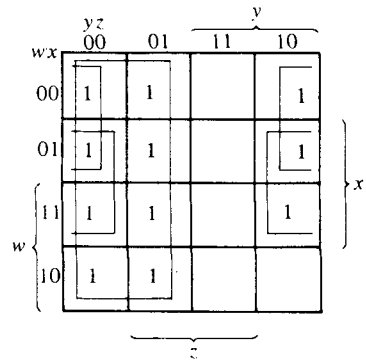


Figura 3-9 Mapa del Ejemplo 3-5; $F(w, x, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = y' + w'z' + xz'$

yor sea el número de cuadrados combinados, menor será el número de literales en el término. En este ejemplo, los dos unos superiores a la derecha se combinan con los dos unos superiores a la izquierda para dar el término $w'z'$. Nótese que es permisible usar el mismo cuadrado más de una vez. Queda entonces un cuadrado marcado con 1 en la tercera fila y cuarta columna (cuadrado 1110). En vez de tomar este cuadrado solo (lo cual dará un término de cuatro literales) se combina con cuadrados ya usados para formar una área de cuatro cuadrados. Estos cuadrados comprenden las dos filas del medio y las dos columnas de los extremos para dar el término xz'. La función simplificada es:

$$F = y' + w'z' + xz'$$

EJEMPLO 3-6: Simplificar la función de Boole:

$$F = A'B'C' + B'CD' + A'BCD' + AB'C'$$

El área, en el mapa, cubierta por esta función consiste en los cuadrados marcados con unos en la Figura 3-10. Esta función tiene cuatro variables y como se ha expresado consiste en tres términos, cada uno con tres literales y un término de cuatro literales. Cada término de tres literales se representa en el mapa por dos cuadrados. Por ejemplo, $A'B'C'$ se representa por los cuadrados 0000 y 0001. La función puede simplificarse en el mapa tomando los unos de las cuatro esquinas para formar el término $B'D'$. Esto es posible porque estos cuatro cuadrados son adyacentes cuando el mapa se dibuja en una superficie con los bordes superior e inferior, izquierdo y derecho tocándose entre sí. Los dos unos de mano izquierda en la fila superior se combinan con los dos unos en la fila inferior para dar el término $B'C'$. El 1 restante puede combinarse en una área de dos cuadrados para dar el término $A'C'D'$. La función simplificada es:

$$F = B'D' + B'C' + A'CD'$$

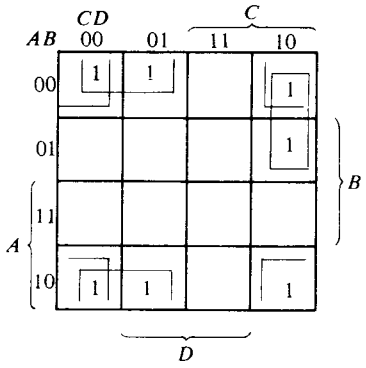


Figura 3-10 Mapa del Ejemplo 3-6; $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

3-4 MAPAS DE CINCO Y SEIS VARIABLES

Los mapas de más de cuatro variables no son simples de usar. El número de cuadrados se hace muy grande y la geometría de combinar cuadrados adyacentes se complica. El número de cuadrados es siempre igual al número de términos mínimos. Para mapas de cinco variables se necesitan 32 cuadrados y para seis variables se necesitan 64 cuadrados. Mapas de siete variables en adelante necesitan muchos cuadrados y son muy imprácticos de usar. En las Figuras 3-11 y 3-12 se muestran los mapas para cinco y seis variables respectivamente. Las columnas y filas se numeran de la misma forma que la secuencia del código reflejado. El término mínimo asignado a cada cuadrado se lee de esos números. De esta manera el cuadrado en la tercera fila (11) y la segunda columna (001) en el mapa para cinco variables se numera 11001 y es equivalente al decimal 25. Por tanto, este cuadrado representa el término mínimo m_{25}. El símbolo de letra de cada variable se marca abarcando aquellos cuadrados donde el valor del bit correspondiente al número del código reflejado es 1. Por ejemplo, en

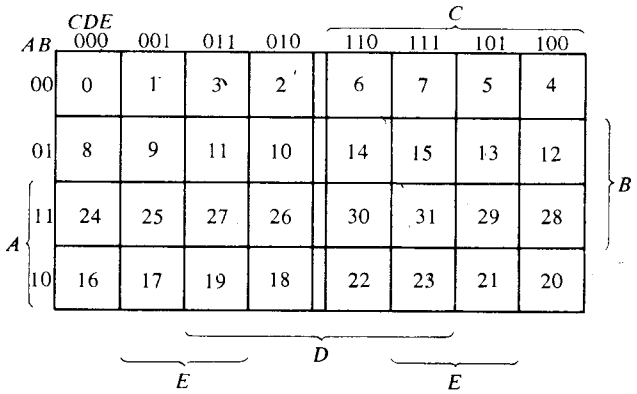


Figura 3-11 Mapa de cinco variables

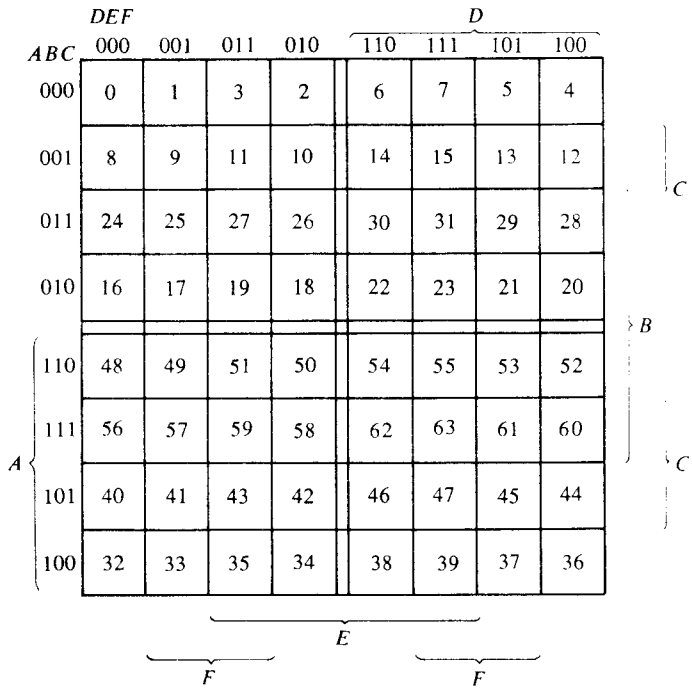


Figura 3-12 Mapa de seis variables

el mapa de cinco variables, la variable *A* es un 1 en las últimas dos filas y *B* es un 1 en las dos filas del medio. Los números reflejados en las columnas muestran la variable *C* con un 1 en las cuatro columnas de la extrema derecha, la variable *D* con un 1 en las cuatro columnas del medio y los unos para la variable *E*, no adyacentes físicamente, se dividen en dos partes. La asignación de las variables en un mapa de seis variables se determina de manera similar.

La definición de los cuadrados adyacentes para los mapas de las Figuras 3-11 y 3-12 deben modificarse de nuevo para tener en consideración el hecho de que algunas variables están divididas en dos partes. Debe pensarse que el mapa de cinco variables consiste en dos mapas de cuatro variables y el mapa de seis variables consiste en cuatro mapas de cuatro variables. Cada uno de estos mapas de cuatro variables se reconocen por las líneas dobles en el centro del mapa; cada uno de ellos conserva la cercanía definida cuando se toma individualmente. Además, la línea doble del centro debe ser considerada como el centro de un libro con cada mitad del mapa como una página. Cuando se cierra el libro, los dos cuadrados adyacentes coinciden uno sobre el otro. En otras palabras, la línea doble del centro actúa como un espejo ya que cada cuadrado es adyacente, no solamente con sus cuatro cuadrados vecinos, sino con su imagen de espejo. Por ejemplo, el término mínimo 31 en el mapa de 5 variables es adyacente a los términos mínimos 30, 15, 29, 23 y 27. El mismo término mínimo en el mapa de seis variables es adyacente a todos esos términos mínimos más el término mínimo 63.

Tabla 3-1 La relación entre el número de cuadrados adyacentes y el número de literales en el término

	Número de cuadrados adyacentes	Número de literales de un término en un mapa de n variables					
k	2^k	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$
0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	4	0	1	2	3	4	5
3	8		0	1	2	3	4
4	16			0	1	2	3
5	32				0	1	2
6	64					0	1

Por inspección y teniendo en cuenta la nueva definición de cuadrados adyacentes, es posible mostrar que cualquier 2^k cuadrados adyacentes para $k = 0, 1, 2, \ldots, n$, en un mapa de n variables, representan una área para un término de $n-k$ literales. Para que la afirmación anterior tenga algún significado, n debe ser mayor que k. Cuando $n = k$ el área total del mapa se combina para dar una función de identidad. La Tabla 3-1 muestra la relación entre el número de cuadrados adyacentes y el número de literales en el término. Por ejemplo, ocho cuadrados adyacentes se combinan en una área del mapa de cinco variables para dar un término de dos literales.

EJEMPLO 3-7: Simplificar la función de Boole:

$$F(A, B, C, D, E) = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$$

El mapa de cinco variables de esta función, se muestra en la Figura 3-13. Cada término mínimo se convierte a un número binario equivalente y los unos se marcan en sus cuadrados correspondientes. Es necesario ahora encontrar combinaciones de cuadrados adyacentes que resulten en la mayor área posible. Los cuatro cuadrados en el centro del mapa de la mitad derecha se reflejan a través de la línea doble y se combinan con los cuatro cuadrados en el centro del mapa de la mitad izquierda, para dar ocho cuadrados adyacentes permisibles equivalentes al término BE. Los dos unos en la fila inferior son el reflejo entre sí con respecto a la línea del centro. Combinándolos con los otros dos cuadrados adyacentes, se obtiene el término $AD'E$. Los cuatro unos en la fila superior son todos adyacentes y pueden ser combinados para dar el término $A'B'E'$. La función simplificada es:

$$F = BE + AD'E + A'B'E'$$

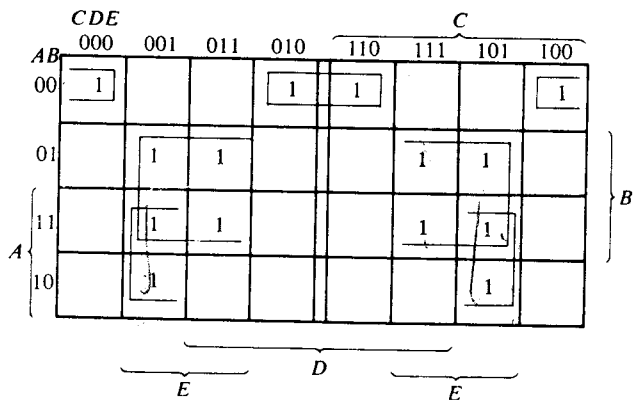


Figura 3-13 Mapa del Ejemplo 3-7; $F(A, B, C, D, E) = \Sigma(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31) = BE + AD'E + A'B'E'$

3-5 SIMPLIFICACION DE UN PRODUCTO DE SUMAS

Las funciones de Boole minimizadas, derivadas del mapa en los ejemplos anteriores fueron expresadas en la forma de suma de productos. Con una pequeña modificación se puede obtener el producto de sumas.

El procedimiento para obtener una función minimizada en producto de sumas se desprende de las propiedades básicas de las funciones de Boole. Los unos colocados en los cuadrados del mapa representan los términos mínimos de la función. Los términos mínimos no incluidos en la función denotan el complemento de una función y se representan en un mapa por cuadrados no marcados por unos. Si se marcan los cuadrados vacíos con ceros y se combinan en cuadrados adyacentes válidos, se obtiene una expresión simplificada del complemento de la función es decir de F'. El complemento de F' dará de nuevo la función F. Debido al teorema generalizado de De Morgan el producto así obtenido queda automáticamente en la forma de producto de sumas. La mejor manera de mostrar esto es mediante un ejemplo.

EJEMPLO 3-8: Simplificar la siguiente función de Boole en (a) suma de productos y (b) producto de sumas.

$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

Los unos marcados en el mapa de la Figura 3-14 representan todos los términos mínimos de la función. Los cuadrados marcados con ceros representan los términos mínimos no incluidos en F y por tanto denotan el complemento de F. Combinando los cuadrados con unos se obtendrá una función simplificada en suma de productos:

$$\text{(a)} \qquad F = B'D' + B'C' + A'C'D$$

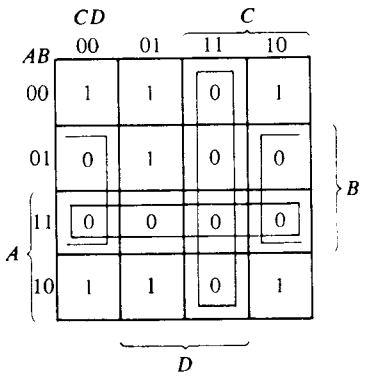


Figura 3-14 Mapa del Ejemplo 3-8; $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$

Si se combinan los cuadrados marcados con ceros, como se muestra en el diagrama, se obtiene la siguiente función simplificada de complemento:

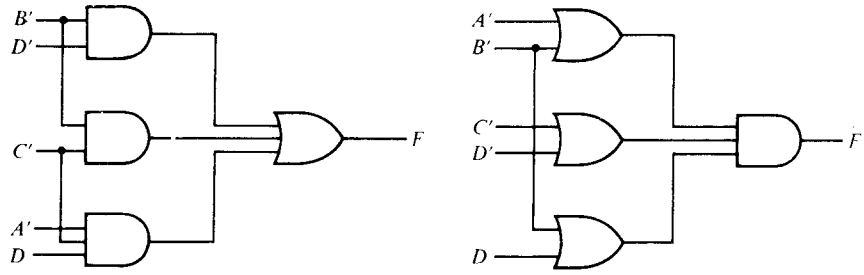
$$F' = AB + CD + BD'$$

Aplicando el teorema de De Morgan (sacándole el dual y complementando cada literal de la manera descrita en la Sección 2-4), se obtiene una función simplificada en producto de sumas:

$$\text{(b)} \quad F = (A' + B')(C' + D')(B' + D)$$

La ejecución de las expresiones simplificadas obtenidas en el Ejemplo 3-8 se muestran en la Figura 3-15. La expresión de la suma de productos se ejecuta en (a) con un grupo de compuertas AND una para cada término AND. Las salidas de las compuertas AND se conectan a las entradas de una compuerta OR. La misma función se ejecuta en (b) en la forma de producto de sumas con un grupo de compuertas OR, una para cada término OR. Las salidas de las compuertas OR se conectan a las entradas de una compuerta AND sencilla. En cada caso se asume que las variables de entrada llegan en forma de complemento de tal manera que no se necesitan inversores. El patrón de configuración establecido en la Figura 3-15 es la forma general por medio de la cual se ejecuta cualquier función de Boole.Una vez expresada en una de las formas normalizadas las compuertas AND se conectan a una compuerta OR en el caso de suma de productos; las compuertas OR se conectan a una sola compuerta AND en el caso de producto de sumas. Cualquiera de las dos configuraciones forman dos niveles de compuertas. Así, la ejecución de una función en la forma normalizada se dice que es una ejecución de dos niveles.

El Ejemplo 3-8 muestra el procedimiento para obtener la simplificación del producto de sumas cuando la función se expresa originalmente en la suma de términos mínimos de la forma canónica. El procedimiento es válido cuando la función se expresa originalmente en el producto de



(a) $F = B'D' + B'C' + A'C'D$ (b) $F = (A' + B')(C' + D')(B' + D)$

Figura 3-15 Ejecución con compuertas de la función del Ejemplo 3-8

Tabla 3-2 Tabla de verdad de la función F

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

términos máximos de la forma canónica. Considérese por ejemplo la tabla de verdad que define la función F en la Tabla 3-2. En suma de términos mínimos esta función se expresa así:

$$F(x, y, z) = \Sigma(1, 3, 4, 6)$$

Como producto de términos máximos se expresa así:

$$F(x, y, z) = \Pi(0, 2, 5, 7)$$

En otras palabras los unos de la función representan los términos mínimos y los ceros representan los términos máximos. El mapa de esta función se dibuja en la Figura 3-16. Se puede simplificar esta función marcando

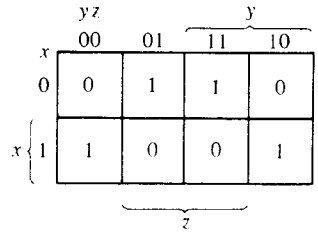


Figura 3-16 Mapa de la función de la Tabla 3-2

primero los unos para cada término mínimo en que la función sea 1. Los cuadrados restantes se marcan como ceros. Si por otra parte se da inicialmente el producto de términos máximos se puede comenzar marcando ceros en aquellos cuadrados que comprende la función; los cuadrados restantes se marcan con unos. Una vez que se hayan marcado los unos y los ceros, la función puede ser simplificada en cualquiera de las dos formas normalizadas. Para la suma de productos se combinan los unos para obtener:

$$F = x'z + xz'$$

Para el producto de sumas se combinan los ceros para obtener la función simplificada del complemento:

$$F' = xz + x'z'$$

lo cual muestra que la función OR-exclusiva es el complemento de la función de equivalencia (Sección 2-6). Tomando el complemento de F' se obtiene la función simplificada en producto de sumas:

$$F = (x' + z')(x + z)$$

Para colocar una función expresada en producto de sumas en el mapa, se saca el complemento de la función y de ella se buscan los cuadrados que se van a marcar con ceros. Por ejemplo, la función:

$$F = (A' + B' + C)(B + D)$$

puede colocarse en el mapa obteniendo primero su complemento:

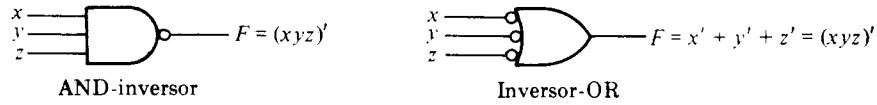
$$F' = ABC' + B'D'$$

para luego marcar con ceros los cuadrados que representan los términos mínimos de F'. Los cuadrados restantes se marcan con unos.

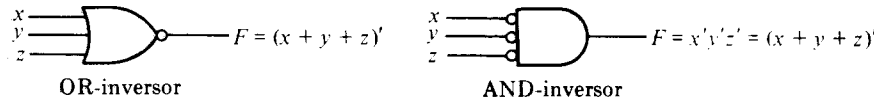
3-6 EJECUCION CON NAND Y NOR

Los circuitos digitales se construyen más frecuentemente con compuertas NAND y NOR que con compuertas AND y OR. Las compuertas NAND y NOR son más fáciles de fabricar con compuertas electrónicas y son las compuertas básicas usadas en todas las familias de CI lógico digitales. Debido a la importancia de las compuertas NAND y NOR en el diseño de circuitos digitales se han desarrollado reglas y procedimientos para la conversión de funciones de Boole en términos de AND, OR y NOT a diagramas lógicos equivalentes en NAND y NOR. El procedimiento para la ejecución en dos niveles se presenta en esta sección. La ejecución en multiniveles se discutirá en la Sección 4-7.

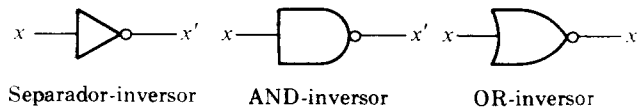
Para facilitar la conversión a lógica NAND y NOR es conveniente definir otros dos símbolos gráficos para estas compuertas. En la Figura 3-17(a) se muestran dos símbolos equivalentes para la compuerta NAND. El símbolo AND inversor ha sido definido precisamente y consiste en un símbolo gráfico AND seguido de un pequeño círculo. En vez de lo anterior es posible



(a) Dos símbolos gráficos para la compuerta NAND



(b) Dos símbolos gráficos para la compuerta NOR



(c) Tres símbolos gráficos para un inversor

Figura 3-17 Símbolos gráficos para las compuertas NAND y NOR

representar una compuerta NAND por medio de un símbolo gráfico OR precedido de pequeños círculos en todas las entradas. El símbolo inversor OR para la compuerta NAND se deduce a partir del teorema de De Morgan y de la convención de que pequeños círculos denotan complementación.

De manera similar, hay dos símbolos gráficos para la compuerta NOR como se muestra en la Figura 3-17(b). El inversor OR es el símbolo convencional. El inversor AND es una alternativa conveniente que utiliza el teorema de De Morgan y la convención de pequeños círculos en las entradas que denotan complementación.

Una compuerta NAND o NOR de una entrada se comporta como un inversor. Como consecuencia una compuerta inversor puede definirse de tres maneras diferentes como se muestra en la Figura 3-17(c). Los círculos pequeños en todos los símbolos de inversor pueden trasferirse al terminal de entrada sin cambiar la lógica de la compuerta.

Se debe resaltar que los símbolos alternos para las compuertas NAND y NOR deben dibujarse con pequeños triángulos en todas las terminales de entrada en vez de los círculos. Un pequeño triángulo es un indicador de la polaridad de la lógica negativa (ver Sección 2-8 y Figura 2-11). Con pequeños triángulos en los terminales de entrada, el símbolo gráfico denota una polaridad de lógica negativa para las entradas, pero la salida de la compuerta (un triángulo) debe tener una asignación de lógica positiva. En este libro, se prefiere usar la lógica positiva y emplear pequeños círculos cuando sea necesario con el fin de denotar complementación.

Ejecución con NAND

La ejecución de una función de Boole con compuertas NAND requieren que la función sea simplificada en la forma de suma de productos. Para ver la

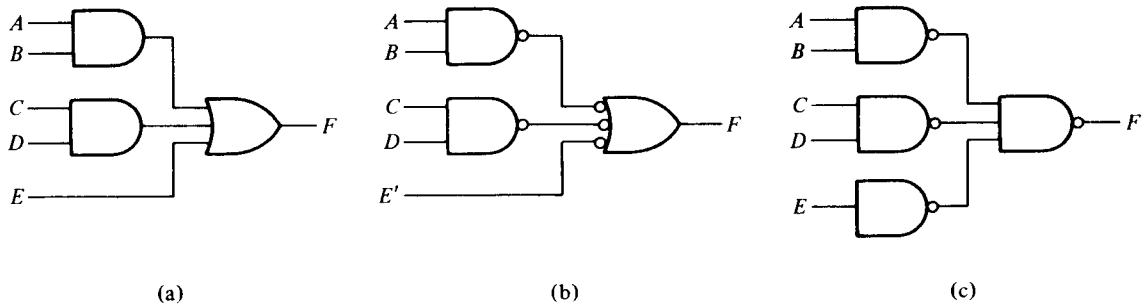


Figura 3-18 Tres maneras de ejecutar $F = AB + CD + E$

relación entre una expresión de suma de productos y su ejecución equivalente en NAND, considérense los diagramas de lógica dibujados en la Figura 3-18. Todos los tres diagramas son equivalentes y ejecutan la función:

$$F = AB + CD + E$$

La función se ejecuta en la Figura 3-18(a) en la forma de suma de productos con compuertas OR y AND. En (b) las compuertas AND se remplazan por compuertas NAND y la compuerta OR se remplaza por la compuerta NAND con un símbolo inversor OR. La variable E por sí sola se complementa y se aplica a la compuerta inversor OR del segundo nivel. Se debe tener en cuenta que un pequeño círculo denota complementación. Así, dos círculos en la misma línea representan doble complementación y ambos pueden anularse. El complemento de E pasa por un pequeño círculo lo cual complementa la variable de una vez más para producir el valor normal de E. Quitando los círculos pequeños en las compuertas de la Figura 3-18(b) se produce el circuito en (a). Así, los dos diagramas ejecutan la misma función y son equivalentes.

En la Figura 3-18(c), la compuerta NAND de salida se puede redibujar con su símbolo convencional. La compuerta NAND de una sola entrada complementa la variable E. Es posible quitar este inversor y aplicar E' directamente a la entrada de la compuerta NAND de segundo nivel. El diagrama en (c) es equivalente al de (b) el cual es equivalente a su turno al diagrama (a). Las compuertas AND y OR han sido cambiadas a compuertas NAND con una sola variable E. Cuando se dibujan los diagramas en lógica NAND son aceptables (b) o (c). El diagrama de la figura (b), sin embargo, representa una relación más directa a la expresión de Boole que ejecuta.

La ejecución con NAND en la Figura 3-18(c) puede verificarse algebraicamente. La función NAND que se ejecuta puede ser convertida fácilmente a una forma de suma de productos mediante el uso del teorema de De Morgan.

$$F = [(AB)' \cdot (CD)' \cdot E']' = AB + CD + E$$

De la trasformación mostrada en la Figura 3-18 se concluye que la función de Boole puede ejecutarse con dos niveles de compuertas NAND. La regla para obtener el diagrama de lógica NAND a partir de una función de Boole es de la siguiente manera:

1. Simplificar la función de Boole y expresarla en suma de productos.
2. Dibujar una compuerta NAND por cada término del producto de la función que tenga por lo menos dos literales. Las entradas a cada compuerta NAND son los literales del término. Lo anterior constituye un grupo de compuertas de primer nivel.
3. Dibujar una compuerta NAND en el segundo nivel, (usando el símbolo gráfico de inversor AND o inversor OR con las entradas que provienen del primer nivel de compuertas.
4. Un término con un solo literal requiere un inversor en el primer nivel o ser complementado primero y aplicado como entrada a una compuerta NAND del segundo nivel.

Antes de aplicarse estas reglas a un ejemplo específico, debe mencionarse que hay una segunda forma de ejecutar una función de Boole con compuertas NAND. Recuérdese que si se combinan los ceros en un mapa, se obtiene la expresión simplificada del *complemento* de la función en suma de productos. El complemento de la función puede ejecutarse con dos niveles de compuertas NAND usando las reglas establecidas anteriormente. Si se desea una salida normal, debe ser necesario colocar una NAND de una entrada o compuerta inversor para generar el valor verdadero de la variable de salida. Hay ocasiones cuando el diseñador quiere generar el complemento de la función para las cuales este método es más aconsejable.

EJEMPLO 3-9: Ejecutar la siguiente función con compuertas NAND:

$$F(x, y, z) = \Sigma(0, 6)$$

El primer paso es simplificar la función en la forma de suma de productos. Esto se logra con el mapa mostrado en la Figura 3-19(a). Hay solamente dos unos en el mapa y no pueden combinarse. La función simplificada para este ejemplo en suma de productos es:

$$F = x'y'z' + xyz'$$

La ejecución con NAND con dos niveles se muestra en la Figura 3-19(b). En seguida se trata de simplificar el complemento de la función en suma de productos. Esto se hace combinando los ceros en el mapa:

$$F' = x'y + xy' + z$$

Las compuertas NAND con dos niveles, para generar F', se muestran en la Figura 3-19(c). Si se requiere la salida F, es necesario agregar una compuerta NAND de una sola entrada para invertir la función. Esto dará una ejecución de tres niveles. Se asume que las variables de entrada se pueden obtener en las formas normales y de complemento. Si sólo se obtienen en una forma será necesario colocar inversores en las entradas, lo cual agregaría otro nivel a los circuitos. La compuerta NAND de una sola entrada asociada con la sola variable z puede eleminarse en el caso de que la entrada se cambie a z'.

Ejecución con NOR

La función NOR es el dual de la función NAND. Por esta razón, todos los procedimientos y reglas para la lógica NOR son el dual de los correspondientes procedimientos y reglas desarrolladas para la lógica NAND.

La ejecución de una función de Boole con compuertas NOR requiere que la función se simplifique en la forma de producto de sumas. Una expresión de producto de sumas especifica un grupo de compuertas OR para la

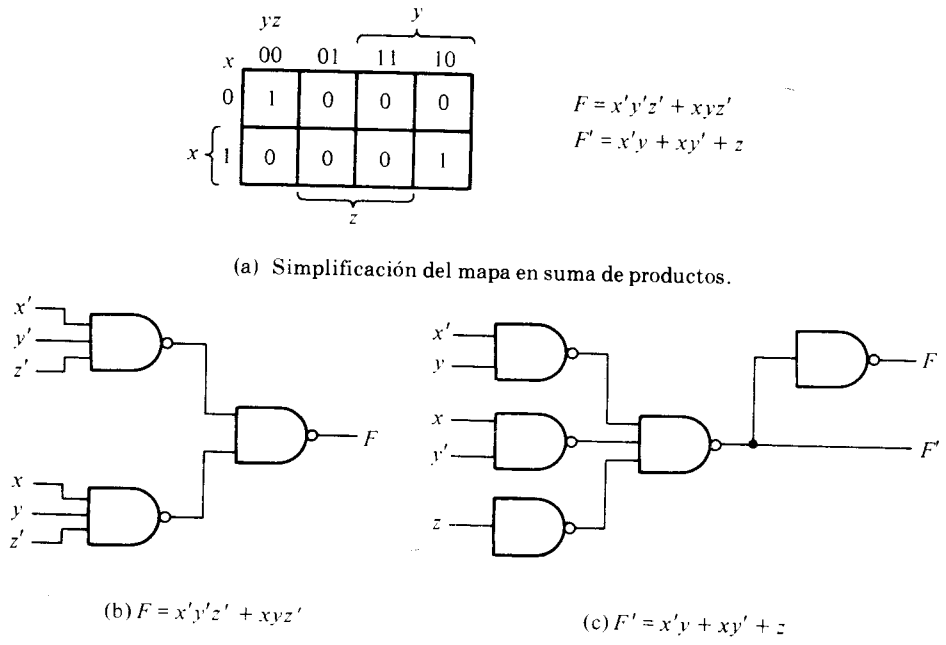


Figura 3-19 Ejecución de la función del Ejemplo 3-9 con compuertas NO-Y

suma de términos, seguida de una compuerta AND para generar el producto. La trasformación del diagrama OR-AND al NOR-NOR se dibuja en la Figura 3-20. Es similar a la trasformación NAND discutida anteriormente excepto que ahora se usa la expresión de suma de productos:

$$F = (A + B)(C + D)E$$

La regla para obtener el diagrama lógico NOR de una función de Boole puede derivarse de esta trasformación. Es similar a la regla NAND de tres pasos con la diferencia de que la expresión simplificada debe estar en producto de sumas y los términos de las compuertas NOR de primer nivel son los términos de suma. Un término con un solo literal requiere una NOR de

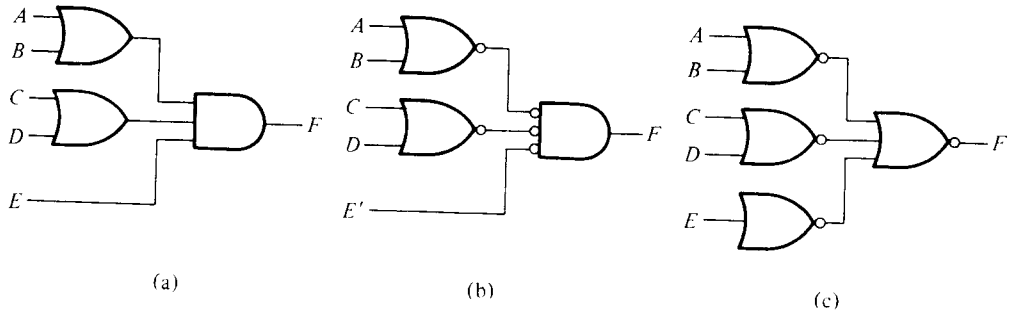


Figura 3-20 Tres maneras de ejecutar $F = (A + B)(C + D)E$

una sola entrada, o compuerta inversora, o ser complementada y aplicada directamente a la compuerta NOR de segundo nivel.

Una segunda manera de ejecutar la función con compuertas NOR podría ser el usar la expresión para el complemento de la función en producto de sumas. Esto dará una ejecución de dos niveles para F' y una ejecución de tres niveles en el caso de necesitarse la salida F normal.

Para obtener el producto de sumas simplificado a partir de un mapa, es necesario combinar los ceros en el mapa y luego complementar la función. Para obtener la expresión en producto de sumas simplificadas para el complemento de la función, es necesario combinar los unos en el mapa y luego complementar la función. El siguiente ejemplo demuestra el procedimiento para una ejecución con NOR.

EJEMPLO 3-10: Ejecutar la función del Ejemplo 3-9 con compuertas NOR.

El mapa de esta función se dibuja en la Figura 3-19(a). Primero, se deben combinar los ceros en el mapa para obtener:

$$F' = x'y + xy' + z$$

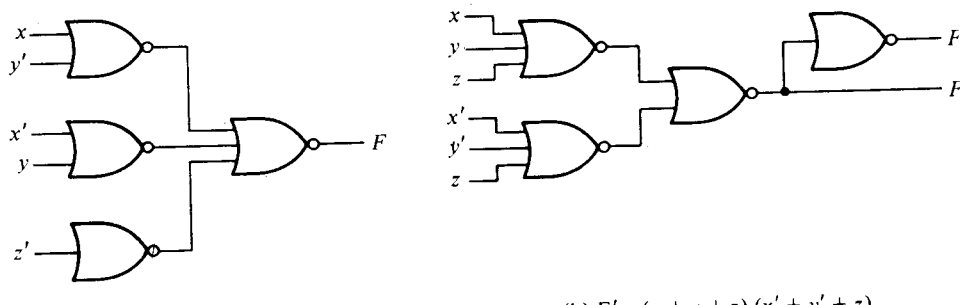
Este es el complemento de la función en suma de productos. Se complementa F' para obtener la función simplificada en producto de sumas de la manera necesaria para la ejecución con NOR:

$$F = (x + y')(x' + y)z'$$

La ejecución de dos niveles con compuertas NOR se muestra en la Figura 3-21(a). El término con un solo literal z' requiere una compuerta NOR de una sola entrada o compuerta inversora. Esta compuerta puede quitarse para aplicar directamente la entrada z a la entrada de la compuerta NOR de segundo nivel.

Una segunda forma de ejecución es posible a partir de la función en producto de sumas. Para este caso combínese primero los unos en el mapa con el fin de obtener:

$$F = x'y'z' + xyz'$$



(a) $F = (x + y')(x' + y)z'$ (b) $F' = (x + y + z)(x' + y' + z)$

Figura 3-21 Ejecución con compuertas NOR

Tabla 3-3 Reglas para la ejecución con NAND y NOR

Caso	Función a simplificar	Forma normal de usar	Como derivarla	Ejecutarse con	Número de niveles de F
(a)	F	Suma de productos	Combine los unos en el mapa	NAND	2
(b)	F'	Suma de productos	Combine los ceros en el mapa	NAND	3
(c)	F	Producto de sumas	Complemente F' en (b)	NOR	2
(d)	F'	Producto de sumas	Complemente F en (a)	NOR	3

Esta es la expresión simplificada en suma de productos. Se complementa esta función para obtener el complemento de la función en producto de sumas que es la forma requerida para la ejecución con NOR:

$$F' = (x + y + z)(x' + y' + z)$$

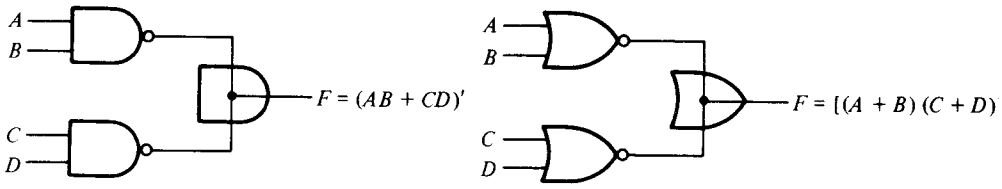
La ejecución de los dos niveles para F' se muestra en la Figura 3-21(b). Si se desea la salida F, esta puede ser generada con un inversor en el tercer nivel.

La Tabla 3-3 resume los procedimientos para la ejecución con NAND y NOR, no se debe olvidar simplificar la función con el fin de reducir el número de compuertas en la ejecución de funciones. Las formas normalizadas obtenidas de los procedimientos de simplificación por mapas se aplican directamente y son muy útiles cuando se está trabajando con lógica NAND o NOR.

3-7 OTRAS EJECUCIONES CON DOS NIVELES

Las clases de compuertas más encontradas a menudo en circuitos integrados son las NAND y NOR. Por esta razón, las ejecuciones de lógica NAND y NOR son las más importantes desde el punto de vista práctico. Algunas compuetas NAND y NOR (pero no todas) permiten la posibilidad de una conexión entre las salidas de las dos compuertas para producir una función lógica específica. Este tipo de lógica se llama lógica de cableado. Por ejemplo, las compuertas NAND TTL de colector abierto, una vez conectadas juntas producen la lógica AND de cableado. (La compuerta TTL de colector abierto se muestra en el Capítulo 13, Figura 13-11). La lógica AND cableada ejecutada con dos compuertas NAND se ilustra en la Figura 3-22(a). La compuerta AND se dibuja con las líneas de entrada atravesando la compuerta hasta el centro para distinguirla de una compuerta comercial. La compuerta AND cableada no es una compuerta física sino solamente un símbolo para designar la función obtenida de la conexión cableada que se indica. La función lógica ejecutada por el circuito de la Figura 3-22(a) es:

$$F = (AB)' \cdot (CD)' = (AB + CD)'$$



(a) AND-cableado en compuertas NAND TTL de colector abierto

(AND-OR INVERSOR)

(b) OR-cableado en compuertas ECL

(OR-AND INVERSOR)

Figura 3-22 Lógica de cableado

y se llama una función AND-OR inversor (o invertida).

De manera similar la salida NOR de las compuertas ECL pueden unirse todas para conformar una función cableada OR. La función lógica ejecutada por el circuito de la Figura 3-22(b) es:

$$F = (A + B)' + (C + D)' = [(A + B)(C + D)]'$$

y se llama función (OR-AND) inversor (o invertida).

Una compuerta de lógica alambrada no produce una compuerta física de segundo nivel ya que se trata solamente de una conexión. Sin embargo, para propósitos de discusión se consideran los circuitos de la Figura 3-22 como ejecuciones de dos niveles. El primer nivel consiste en compuertas NAND (o NOR) y el segundo nivel tiene una compuerta sencilla AND (u OR). La conexión cableada del símbolo gráfico se omitirá en las discusiones subsiguientes.

Formas no degeneradas

Es instructivo desde el punto de vista teórico encontrar cuantas combinaciones de compuertas de dos niveles son posibles. Se consideran cuatro tipos de compuertas: AND, OR, NAND y NOR. Si se asigna un tipo de compuertas para el primer nivel y uno para el segundo se encuentra que existen 16 combinaciones posibles de formas de dos niveles. (El mismo tipo de compuerta puede estar en el primer y segundo niveles como en una ejecución con NAND-NAND). Ocho de estas funciones se les llama formas degeneradas. Esto puede verse de un circuito con compuertas Y en el primer nivel y una compuerta Y en el segundo nivel. La salida del circuito es simplemente la función Y de todas las variables de entrada. Las otras ocho formas *no degeneradas* producen formas de ejecución en suma de productos o producto de sumas. Las ocho formas no degeneradas son:

AND-OR	OR-AND
NAND-NAND	NOR-NOR
NOR-OR	NAND-AND
OR-AND	AND-OR

La primera compuerta de cada una de las formas listadas constituye el primer nivel de la ejecución. La segunda compuerta de la lista es una sola compuerta colocada en el segundo nivel. Nótese que cualquier par de formas de la lista son duales entre sí.

Las formas AND-OR y OR-AND son las dos formas básicas de dos niveles discutidas en la Sección 3-5. Las NAND-NAND y NOR-NOR se introdujeron en la Sección 3-6. Las cuatro formas restantes se investigan en esta sección.

Ejecución con AND-OR invertida

La dos formas NAND-AND y AND-NOR son formas equivalentes y pueden ser tratadas conjuntamente. Ambas realizan la función AND-OR invertida de la manera mostrada en la Figura 3-23. La forma AND-NOR se parece a la forma AND-OR con una inversión hecha por un pequeño círculo a la salida de la compuerta NOR. Esta ejecuta la función:

$$F = (AB + CD + E)'$$

Usando el símbolo gráfico alterno para la compuerta NOR se obtiene el diagrama de la Figura 3-23(b). Nótese que la sola variable E *no* es complementada porque el único cambio hecho es el símbolo gráfico de la compuerta NOR. Se trasladan los círculos del terminal de entrada de las compuertas de segundo nivel a los terminales de salida de las compuertas del primer nivel. Se necesita solamente un inversor para que la sola variable mantenga el círculo. Otra alternativa es quitar el inversor siempre y cuando la entrada E esté complementada. El circuito de la Figura 3-23(c) es una forma NAND-AND, se muestra en la Figura 3-22 con el fin de ejecutar la función AND-OR invertida.

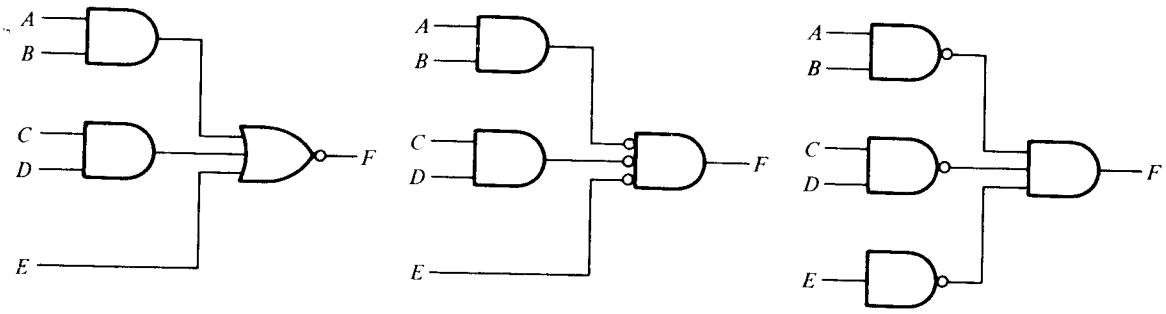
Una ejecución con AND-OR requiere una expresión en suma de productos. La ejecución con AND-OR invertida es similar excepto por la inversión (negado). Por tanto, si el *complemento* de una función se simplifica en suma de productos (combinando los ceros en el mapa), es posible ejecutar F' con la parte AND-OR de la función. Cuando F' pase por la inversión de salida siempre presente, se generará la salida F de la función. Un ejemplo de una ejecución con AND-OR invertida se mostrará más adelante.

Ejecución con OR-AND invertida

Las formas OR-NAND y NOR-OR realizan la función OR-AND invertida como se muestra en la Figura 3-24. La forma OR-NAND se parece a la forma OR-AND excepto por la inversión hecha por el círculo en la compuerta NAND. Ella ejecuta la función:

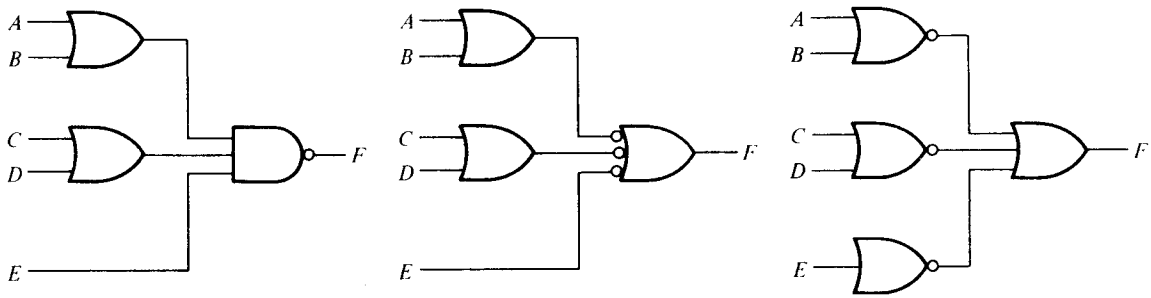
$$F = [(A + B)(C + D)E]'$$

Mediante el uso de un símbolo gráfico alterno para la compuerta NAND se obtiene el diagrama de la Figura 3-24(b). El circuito en (c) se obtiene moviendo los círculos pequeños de las entradas de la compuerta de se-



(a) AND-NOR (b) AND-NOR (c) NAND-AND

Figura 3-23 Circuitos AND-OR invertidos; $F = (AB + CD + E)$



(a) OR-NAND (b) OR-NAND (c) NOR-OR

Figura 3-24 Circuitos OR-AND invertidas; $F = [(A + B)(C + D)E]'$

gundo nivel a las salidas de las compuertas de primer nivel. El circuito de la Figura 3-24(c) en una forma NOR-OR se muestra en la Figura 3-22 para ejecutar la función OR-AND invertida.

La ejecución OR-AND invertida requiere una expresión en producto de sumas. Si el complemento de la función se simplifica en producto de sumas se puede ejecutar F' con la parte OR-AND de la función. Una vez que F' pase por la parte de inversión se obtiene el complemento de F' o sea F a la salida.

Tabla sumario y ejemplo

La Tabla 3-4 resume los procedimientos para la ejecución de funciones de Boole en cualquiera de las cuatro formas de dos niveles. Debido a la parte de INVERSION, en cada caso es conveniente usar la simplificación F' (el complemento) de la función. Cuando se ejecuta F' en una de estas formas se obtiene el complemento de la función en la forma AND-OR u OR-AND. Las cuatro formas de dos niveles invierten esta función dando una salida que es el complemento de F'. Esta última es la salida normal F.

Tabla 3-4 Ejecución con otras formas de dos niveles

Forma equivalente no degenerada		Ejecuta la función	Simplifique F' en	Para obtener una salida de
(a)	(b)*			
AND-NOR	NAND-AND	AND-OR-INVERTIDA	Suma de productos combinando los ceros en el mapa	F
OR-NAND	NOR-OR	OR-AND-INVERTIDA	Producto de sumas combinando los unos en el mapa y luego complementando.	F

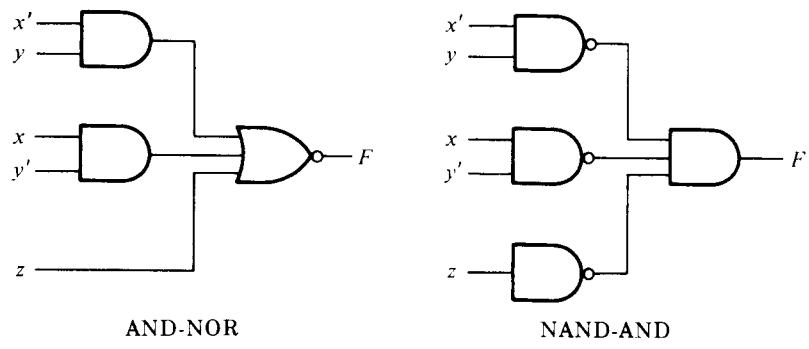
*La forma (b) requiere una compuerta NAND de una entrada a una NOR (inversor) para el término de un solo literal.

EJEMPLO 3-11: Ejecútese la función de la Figura 319(a) con las cuatro formas de dos niveles listados en la Tabla 3-4. El complemento de la función se simplifica en suma de productos combinando los ceros del mapa:

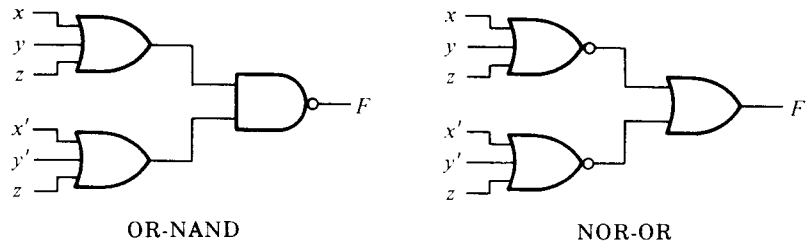
$$F' = x'y + xy' + z$$

La salida normal de esta función puede ser expresada como:

$$F = (x'y + xy' + z)'$$



(a) $F = (x'y + xy' + z)'$



(b) $F = [(x + y + z)(x' + y' + z)]'$

Figura 3-25 Otras ejecuciones de dos niveles

la cual está en la forma AND-OR invertida. Las ejecuciones con AND-NOR y NAND-AND se muestran en la Figura 3-25(a). Nótese que una NAND de una entrada o compuerta inversora se necesita para la ejecución con NAND-AND, pero no en el caso AND-OR. El inversor puede eliminarse si se aplica una variable de entrada z' en vez de z.

Las formas OR-AND invertida requieren una expresión simplificada del complemento de las funciones en producto de sumas. Para obtener esta expresión se deben combinar los unos en el mapa:

$$F = x'y'z' + xyz'$$

En seguida se toma el complemento de la función:

$$F' = (x + y + z)(x' + y' + z)$$

La salida normal F puede ahora expresarse en la forma:

$$F = [(x + y + z)(x' + y' + z)]'$$

la cual está en la forma OR-AND invertida. A partir de esta expresión se puede ejecutar la función en las formas OR-NAND y NOR-OR como se muestra en la Figura 3-25(b).

3-8 CONDICIONES DE NO IMPORTA

Los unos y ceros en el mapa significan la combinación de variables que hacen la función igual a 1 ó 0 respectivamente. Las combinaciones se obtienen comúnmente de una tabla de verdad que lista las condiciones bajo las cuales la función es 1. Se asume que la función sea igual a cero bajo cualquier otra condición. Esta suposición no es siempre verdadera ya que hay aplicaciones donde ciertas combinaciones de variables de entrada nunca ocurren. Un código decimal de cuatro bits, por ejemplo, tiene seis combinaciones que no se usan. Cualquier circuito digital que use este código, opera bajo la suposición de que esas combinaciones no usadas nunca ocurren, siempre y cuando el sistema esté trabajando adecuadamente. Como resultado, no importa lo que sea la salida de la función para estas combinaciones de variables ya que se garantiza que nunca ocurrirán. Estas condiciones de no importa pueden usarse en un mapa para lograr una mejor simplificación de la función.

Se puede hacer énfasis en que la combinación de no importa no puede ser marcada con un 1 en el mapa ya que ella implica que la función sea 1 para esa combinación de entrada. De la misma manera colocar un cero requiere que la función sea cero. Para diferenciar las condiciones de no importa de los unos y ceros se usará una X.

Cuando se escogen cuadrados adyacentes, para simplificar la función en el mapa, se asume que la X sea 1 ó 0 según lo que produzca la expresión más simple. Además, no se necesita usar la X si esta no contribuye al cubrimiento de una área mayor. En cada caso, la alternativa depende solamente de la simplificación que se puede lograr.

EJEMPLO 3-12: Simplificar la función de Boole:

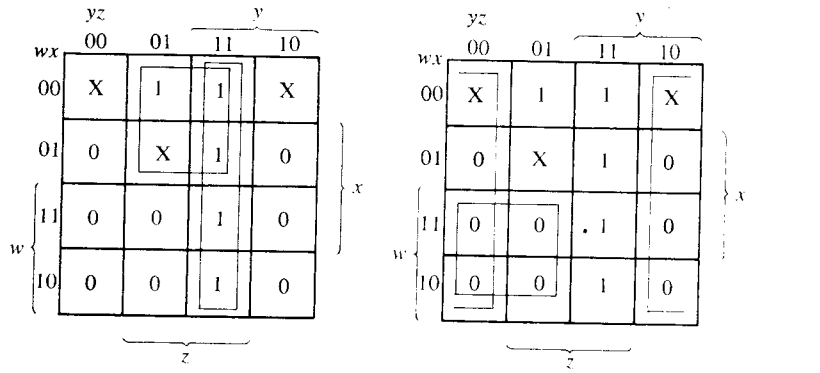
$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

y las condiciones de no importa:

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$

Los términos mínimos de F son las combinaciones de variables que hacen la función igual a 1. Los términos mínimos de d son las combinaciones de no importa que se conoce que nunca ocurren. La minimización se muestra en la Figura 3-26. Los términos mínimos de F se marcan con unos y aquellos de d se marcan con una X y los cuadrados restantes se llenan con ceros. En (a) los unos y las X se combinan de una forma conveniente tal que se abarque el mayor número de cuadrados adyacentes. No es necesario incluir todas o algunas de las X sino aquellas que sean útiles para la simplificación de un término. Una combinación que da una función mínima incluye una X y deja dos por fuera. Esto dará como resultado una función simplificada en suma de productos.

$$F = w'z + yz$$



(a) Combinando unos y X $F = w'z + yz$ (b) Combinando ceros y X $F = z(w' + y)$

Figura 3-26 Ejemplo con condiciones de no importa

En (b), los ceros se combinan con cualquier X conveniente para simplificar el complemento de la función. Los mejores resultados se obtienen si se incluyen las X de la manera mostrada. La función complementada se simplifica para dar:

$$F' = z' + wy'$$

Complementándola de nuevo se obtiene una función simplificada en producto de sumas:

$$F = z(w' + y)$$

Las dos expresiones obtenidas en el Ejemplo 3-12 dan dos funciones, las cuales se pueden demostrar como algebraicamente iguales. Este no es siempre el caso cuando intervienen condiciones de no importa. De hecho, si una X se usa como 1, cuando se combinan los unos o como 0 cuando se combinan los ceros, las dos funciones resultantes no producirán respuestas iguales algebraicamente. La selección de la condición de no importa como 1 en el primer caso y como 0 en el segundo, resulta en expresiones de diferentes términos mínimos y por tanto en diferentes funciones. Esto puede verse del Ejemplo 3-12. En la solución del mismo la X escogida como 1 no se escogió como cero. Ahora, si en la Figura 3-26(a) se escoge el término $w'z'$ en vez de $w'z$ se obtiene de todas formas una función minimizada:

$$F = w'x' + yz$$

Pero que no es algebraicamente igual a la obtenida en producto de sumas porque la misma X se usa como 1 en la primera minimización y como cero en la segunda.

Este ejemplo demuestra también que una expresión con un mínimo de literales no es necesariamente única. Algunas veces el diseñador se encuentra con una alternativa entre dos términos con un número igual de literales, tal que la escogencia de cualquiera resulta en una expresión minimizada.

3-9 EL METODO DEL TABULADO

El método del mapa para simplificación es conveniente siempre y cuando el número de variables no exceda de cinco o seis. A medida que el número de variables aumenta el número excesivo de cuadrados impide una selección razonable de cuadrados adyacentes. La desventaja obvia del mapa es esencialmente el procedimiento de prueba y error que depende de la habilidad del usuario humano para reconocer ciertos patrones. Para funciones de seis o más variables es muy difícil estar seguro que realmente se hizo la mejor selección.

El método del tabulado elimina la anterior dificultad. Este se trata de un procedimiento específico paso a paso que se garantiza para producir una expresión de forma normalizada y simplificada. Este se puede aplicar a problemas con muchas variables y tiene la ventaja de ser adecuado para cómputos con máquina. Sin embargo es un poco tedioso para uso humano y propenso a errores debido a un proceso rutinario y monótono. El método del tabulado fue formulado primero por Quine (3) y más tarde mejorado por McCluskey.

El método de simplificación consiste en dos partes. La primera es encontrar mediante una búsqueda muy completa de todos los términos candidatos de inclusión en la función simplificada. Estos términos se llaman *primeros-implicados*. La segunda operación es escoger entre los primeros implicados aquellas que dan una expresión con el menor número de literales.

3-10 DETERMINACION DE LOS PRIMEROS IMPLICADOS*

El punto de partida del método del tabulado es la lista de términos mínimos que especifican la función. La primera operación de tabulado es buscar los primeros implicados para usarlos en el proceso de apareamiento. Este proceso compara cada término mínimo con cada uno de los restantes términos mínimos. Si dos términos mínimos difieren en solamente una variable, esa variable se elimina para encontrar un solo término con un literal menos. Este proceso se repite para cada término mínimo hasta que se complete el proceso completo de búsqueda. El ciclo del proceso de apareamiento se repite para aquellos términos nuevos encontrados. Se continúa con el tercer y subsiguientes ciclos hasta el paso por un ciclo no produzca nuevas eliminaciones de literales. Los términos restantes y todos los términos que no se aparearon durante el proceso, constituyen los primeros implicados. El método del tabulado se ilustra por medio del ejemplo siguiente:

EJEMPLO 3-13: Simplificar la siguiente función de Boole usando el método del tabulado:

$$F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$$

* Esta sección y la siguiente pueden ser omitidas sin perder continuidad.

Paso 1: Agrupar la representación binaria de los términos mínimos de acuerdo al número de unos contenido de la manera mostrada en la Tabla 3-5 columna (a). Esto se hace agrupando los términos mínimos en cinco secciones separadas por líneas horizontales. La primera sección contiene el número sin unos en él. La segunda sección contiene aquellos números que tienen solamente un uno. La tercera, cuarta y quinta sección contienen aquellos números binarios con dos, tres y cuatro unos respectivamente. Los decimales equivalentes de los términos mínimos se colocan a todo lo largo para identificación.

Paso 2: Cualquier par de términos mínimos que difieren entre sí solamente por una variable, se pueden combinar y las variables no apareadas eliminar. Dos números de término mínimo caen dentro de esta categoría si ambos tienen el mismo valor de bit en todas las posiciones excepto en una. Los términos mínimos en una sección se comparan con aquellos de la siguiente en adelante ya que dos términos que se diferencian en más de un bit no se pueden aparear. El término mínimo de la primera sección se compara con cada uno de los tres términos mínimos de la segunda sección. Si hay dos términos iguales en todas las posiciones excepto en una, se marcan a la derecha de ambos términos mínimos para demostrar que han sido usados. El término resultante, conjuntamente con los equivalentes decimales, se lista en la columna (b) de la tabla. La variable eliminada durante el proceso de apareamiento se remplaza por un guión en su posición original. En

Tabla 3-5 Determinación de los primeros implicados para el Ejemplo 3-13

(a)		(b)		(c)	
	w x y z		*w x y z*		*w x y z*
0	0 0 0 0 √	0, 1	0 0 0 −	0, 2, 8, 10	− 0 − 0
		0, 2	0 0 − 0 √	0, 8, 2, 10	− 0 − 0
1	0 0 0 1 √	0, 8	− 0 0 0 √	10, 11, 14, 15	1 − 1 −
2	0 0 1 0 √			10, 14, 11, 15	1 − 1 −
8	1 0 0 0 √	2, 10	− 0 1 0 √		
		8, 10	1 0 − 0 √		
10	1 0 1 0 √				
		10, 11	1 0 1 − √		
11	1 0 1 1 √	10, 14	1 − 1 0 √		
14	1 1 1 0 √				
		11, 15	1 − 1 1 √		
15	1 1 1 1 √	14, 15	1 1 1 − √		

este caso m_0 (0000) se combina con m_1 (0001) para formar (000-). Esta combinación es equivalente a la operación algebraica:

$$m_0 + m_1 = w'x'y'z' + w'x'y'z = w'x'y'$$

El término mínimo m_0 se combina con m_2 para formar (00-0) y con m_8 para formar (-000). El resultado de esta comparación se coloca en la primera sección de la columna (b). Los términos mínimos de las secciones dos y tres de la columna (a) se comparan en seguida para producir los términos listados en la segunda sección de la columna (b). Todas las otras secciones de (a) se comparan de manera similar y las secciones subsecuentes se forman en (b). Este proceso de comparación dará como resultado cuatro secciones de (b).

Paso 3: Los términos de la columna (b) tienen solamente tres variables. Un 1 debajo de la variable significa que no es tildada, un 0 significa que es tildada y un guión significa que no se incluye en el término. El proceso de búsqueda y comparación se repite para los términos en la columna (b) para formar los dos términos variables de la columna (c). De nuevo, los términos en cada sección necesitan compararse solamente si tienen guiones en la misma posición. Nótese que el término (000-) no se aparea con cualquier otro término. Por consiguiente, este no tendrá marca a su derecha. Los equivalentes decimales se escriben a mano derecha de cada entrada para propósitos de identificación. El proceso de comparación debe llevarse a cabo de nuevo en la columna (c) y en las columnas subsiguientes siempre y cuando se consiga el apareamiento adecuado. En el ejemplo presente, la operación para en la tercera columna.

Paso 4: Los términos no marcados en la tabla forman los primeros implicados. En este ejemplo tenemos el término $w'x'y'$ (000-) en la columna (b) y los términos $x'z'$ (-0-0) y wy (1-1-) en la columna (c). Nótese que cada término de la columna (c) aparece dos veces en la tabla y cuando el término forme un primer implicado es innecesario usar el mismo término dos veces. La suma de los primeros implicados dará una expresión simplificada de la función. Esto es debido a que cada término marcado en la tabla se ha tenido en cuenta para la entrada de un término más sencillo en la columna subsecuente. Así, las entradas no marcadas (primeros-implicados) constituyen los términos dejados para formular la función. Para el ejemplo presente, la suma de los primeros implicados dará la función minimizada en suma de productos:

$$F = w'x'y' + x'z' + wy$$

Vale la pena comparar la anterior respuesta con la obtenida mediante el método del mapa. La Figura 3-27 muestra la simplificación por mapa de esta función. Las combinaciones de los cuadrados adyacentes dan los

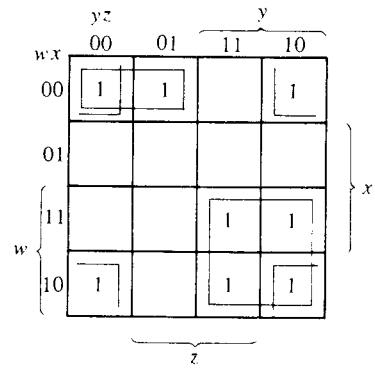


Figura 3-27 Mapa de la función del Ejemplo 3-13; $F = w'x'y' + x'z' + wy$

tres primeros implicados de la función. La suma de estos tres términos es la expresión simplificada en suma de productos.

Es importante señalar que el Ejemplo 3-13 fue escogido a propósito para dar una función simplificada a partir de una suma de primeros implicados. En la mayoría de los casos la suma de los primeros implicados no necesariamente forman la expresión con el número mínimo de términos. Esto se demuestra en el Ejemplo 3-14.

La tediosa manipulación que se debe hacer cuando se usa el método del tabulado se reduce si la comparación se hace con números decimales en vez de binarios. Se mostrará ahora un método que usa la resta de números decimales en vez de comparar y aparear números binarios. Nótese que cada 1 en un número binario representa el coeficiente multiplicado por una potencia de 2. Cuando dos términos mínimos son iguales en todas las posiciones excepto en una, el término mínimo con el 1 extra debe ser más grande, que el número del otro término mínimo, en una potencia de 2. Por tanto, dos términos mínimos se pueden cambiar si el número del primer término mínimo difiere por una potencia de 2 de un segundo número mayor de la siguiente sección inferior de la tabla. Se ilustrará este procedimiento repitiendo el Ejemplo 3-13.

Como se muestra en la Tabla 3-6 columna (a), los términos mínimos se arreglan en secciones como se hizo anteriormente excepto que se listan solamente los decimales equivalentes a los términos mínimos. El proceso de comparar los términos mínimos es como sigue: inspecciónese todo par de números decimales en secciones adyacentes de la tabla. Si el número de la sección inferior es *mayor* que el número de la sección superior por una potencia de 2 (por ejemplo 1, 2, 4, 8, 16, etc.) márquese ambos números para demostrar que han sido usados y escríbalos en la columna (b). El par de números trasferidos a la columna (b) incluyen un tercer número en paréntesis que designa la potencia de 2 por la cual difieren los números. El número en paréntesis dice la posición del guión en la notación binaria. El resultado de la comparación de la columna (a) se muestra en la columna (b).

La comparación entre secciones adyacentes en la columna (b) se realiza de manera similar, excepto que solamente se comparan aquellos tér-

Tabla 3-6 Determinación de los primeros-implicados del Ejemplo 3-13 con notación decimal

(a)		(b)		(c)	
0	√	0, 1 (1)		0, 2, 8, 10	(2, 8)
		0, 2 (2)	√	0, 2, 8, 10	(2, 8)
1	√	0, 8 (8)	√		
2	√			10, 11, 14, 15	(1, 4)
8	√	2, 10 (8)	√	10, 11, 14, 15	(1, 4)
		8, 10 (2)	√		
10	√				
		10, 11 (1)	√		
11	√	10, 14 (4)	√		
14	√				
		11, 15 (4)	√		
15	√	14, 15 (1)	√		

minos con el mismo número en paréntesis. El par de números en una sección debe diferir por una potencia de 2 del par de números en la siguiente sección. Y los números en la sección inmediatamente inferior deben ser *mayores* para poder lograr la combinación. En la columna (c) escríbase todos los cuatro números decimales, con los dos números en paréntesis como indicadores de la posición de los guiones. Una comparación de las Tablas 3-5 y 3-6 podría ser útil para comprender las derivaciones de la Tabla 3-6.

Los primeros implicados son aquellos términos no marcados en la tabla. Son los mismos que los encontrados anteriormente excepto que están dados en notación decimal. Para convertir la notación decimal a binaria conviértase todos los números decimales en el término a binarios y luego colóquese un guión en aquellas posiciones designadas por los números en paréntesis. Así 0,1 (1) se convierte a binario como 0000, 0001; un guión en la primera porción de cada número resultará en (000-). De la misma manera, 0, 2, 8, 10 (2, 8) se convierte a la notación binaria 0000, 0010, 1000 y 1010, y un guión colocado en las posiciones 2 y 8, dará como resultado (-0-0).

EJEMPLO 3-14: Determinar los primeros implicados de la función:

$$F(w, x, y, z) = \Sigma(1, 4, 6, 7, 8, 9, 10, 11, 15)$$

Los números de los términos mínimos se agrupan en secciones de la manera mostrada en la Tabla 3-7 columna (a). El binario equivalente de un término mínimo se incluye con el propósito de contar el número de unos. Los números binarios en la primera sección

Tabla 3-7 Determinación de los primeros implicados del Ejemplo 3-14

(a)			(b)			(c)
0001	1	√	1, 9	(8)		8, 9, 10, 11 (1, 2)
0100	4	√	4, 6	(2)		8, 9, 10, 11 (1, 2)
1000	8	√	8, 9	(1)	√	
			8, 10	(2)	√	
0110	6	√				
1001	9	√	6, 7	(1)		
1010	10	√	9, 11	(2)	√	
			10, 11	(1)	√	
0111	7	√				
1011	11	√	7, 15	(8)		
			11, 15	(4)		
1111	15	√				

Primeros-implicados

Decimal	Binario $w\ x\ y\ z$	Términos
1, 9 (8)	− 0 0 1	$x'y'z$
4, 6 (2)	0 1 − 0	$w'xz'$
6, 7 (1)	0 1 1 −	$w'xy$
7, 15 (8)	− 1 1 1	xyz
11, 15 (4)	1 − 1 1	wyz
8, 9, 10, 11 (1, 2)	1 0 − −	wx'

tienen sólo un uno, en la segunda sección dos unos, etc. Los números de los términos mínimos se comparan por el método decimal y se hacen parejas, si el número de la sección inferior es mayor que aquel de la sección superior. Si el número de la sección inferior es más pequeño que el de la superior no se tiene en cuenta la pareja aunque los dos números difieren por una potencia de 2. La búsqueda minuciosa en la columna (a) dará como resultado los términos de la columna (b), con todos los términos mínimos en la columna (a) marcados. Hay solamente dos parejas de términos en la columna (b) las cuales darán el mismo término de dos literales en la columna (c). Los primeros implicados consisten en todos los términos no marcados en la tabla. La conversión de notación binaria a decimal se muestra en la parte inferior de la tabla. Los primeros implicados encontrados son $x'y'z$, $w'xz'$, $w'xy$, xyz, wyz y wx'.

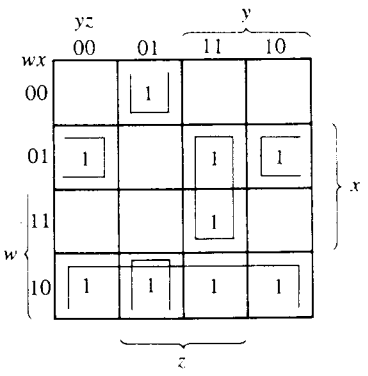


Figura 3-28 Mapa de la función del Ejemplo 3-14;
$F = x'y'z + w'xz' + xyz + wx'$

La suma de todos los primeros implicados, dará una expresión algebraica válida para la función. Sin embargo esta expresión no es necesariamente la que contiene el mínimo número de términos. Esto puede demostrarse inspeccionando el mapa de la función del Ejemplo 3-14. Como se muestra en la Figura 3-28 la función minimizada reconocida es:

$$F = x'y'z + w'xz' + xyz + wx'$$

la cual consiste en la suma de cuatro de los seis primeros implicados derivados del Ejemplo 3-14. El procedimiento de tabulado para la selección de los primeros implicados que dan la función minimizada es el tema de la siguiente sección.

3-11 SELECCION DE LOS PRIMEROS IMPLICADOS

La selección de los primeros implicados que forman la función minimizada se hace a partir de una tabla de primeros implicados. En esta tabla, cada primer implicado se representa en una fila y cada término mínimo en una columna. Se colocan cruces en cada fila para mostrar la composición de los términos mínimos que constituyen los primeros implicados. Un mínimo grupo de primeros implicados se escoge de manera que abarque todos los términos mínimos de la función. Este procedimiento se ilustra en el Ejemplo 3-15.

> ***EJEMPLO 3-15:*** Minimizar la función del Ejemplo 3-14. El tabulado de los primeros implicados para este ejemplo se muestra en la Tabla 3-8. Hay seis filas, una para cada primer implicado (derivado en el Ejemplo 3-14) y nueve columnas que representan cada una un término mínimo de la función. Se colocan cruces en cada fila para indicar los términos mínimos contenidos en el primer implicado de esa fila. Por ejemplo, las dos cruces en la primera fila indican que los términos mínimos 1 y 9 están contenidos en el primer implicado $x'y'z$. Es aconsejable incluir el equivalente deci-

Tabla 3-8 Tabla de primeros-implicados del Ejemplo 3-15

		1	4	6	7	8	9	10	11	15
$\surd x'y'z$	1, 9	X					X			
$\surd w'xz'$	4, 6		X	X						
$w'xy$	6, 7			X	X					
xyz	7, 15				X					X
wyz	11, 15								X	X
$\surd wx'$	8, 9, 10, 11					X	X	X	X	
		$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$	$\surd$	

mal del primer implicado en cada fila y conveniente dar los términos mínimos contenidos en él. Una vez se hayan marcado todas las cruces se procederá a seleccionar un número mínimo de primeros implicados.

La tabla completa de primeros implicados se inspecciona para obtener columnas que contengan solamente una cruz. En este ejemplo hay cuatro términos mínimos cuyas columnas tienen una sola cruz: 1, 4, 8 y 10. El término mínimo 1 está cubierto por el primer implicado $x'y'z$; es decir, la selección del primer implicado $x'y'z$ garantiza que el término mínimo 1 está incluido en la función. De manera similar el término mínimo 4 está cubierto por el primer implicado $w'xz'$ y los términos mínimos 8 y 10 por el primer implicado wx'. Los primeros implicados que cubren los términos mínimos con una sola cruz en su columna se llaman *primeros implicados esenciales*. Para permitir que la expresión final simplificada contenga todos los términos mínimos no queda otra alternativa que incluir los primeros implicados esenciales. Se coloca una marca en la tabla a continuación de los primeros implicados esenciales para indicar que han sido seleccionados.

En seguida se observa cada columna cuyo término mínimo está cubierto por los primeros implicados esenciales seleccionados. Por ejemplo, el primer implicado seleccionado $x'y'z$ cubre los términos mínimos 1 y 9, entonces se coloca una marca en la parte inferior de las columnas. De manera similar, el primer, implicado $w'xz'$ cubre los términos mínimos 4 y 6 y wx' cubre 8, 9, 10 y 11 respectivamente. La inspección de la tabla de primeros implicados cubre todos los términos de la función con excepción de 7 y 15. Estos dos términos mínimos deben ser incluidos por la selección de uno o más primeros implicados. En este ejemplo es claro que el primer implicado xyz cubre ambos términos mínimos y es por tanto el seleccionado. Así se ha encontrado el conjunto mínimo de primeros implicados cuya suma da la función minimizada requerida:

$$F = x'y'z + w'xz' + wx' + xyz$$

Las expresiones simplificadas deducidas en los ejemplos anteriores estaban expresadas en la forma de suma de productos. El método del tabulado puede adaptarse para dar una expresión simplificada en producto de sumas. De la misma manera que en el método del mapa se tiene que comenzar con el complemento de la función tomando los ceros como la lista inicial de términos mínimos. Esta lista contiene aquellos términos mínimos no incluidos en la función original, los cuales son numéricamente iguales a los términos máximos de la función. El proceso de tabulación se lleva a cabo con los ceros de la función para terminar con una expresión simplificada en suma de productos del complemento de la función. Obteniendo de nuevo el complemento se consigue la expresión simplificada en producto de sumas.

Una función con condiciones de no importa puede ser simplificada por el método del tabulado después de una pequeña modificación. Los términos de no importa se incluyen en la lista de los términos mínimos cuando los primeros implicados se determinan. Esto permite la deducción de primeros implicados con el mínimo número de literales. Los términos de no importa no se incluyen en la lista de los términos mínimos cuando se prepara la tabla de los primeros implicados ya que los términos de no importa no tienen que estar cubiertos por los primeros implicados seleccionados.

3-12 OBSERVACIONES CONCLUYENTES

Se introdujeron dos métodos de simplificación de funciones de Boole en este capítulo. El criterio para la simplificación fue el de minimizar el número de literales en expresiones de suma de productos o productos de sumas. Tanto el método del mapa como el de tabulado son tan restringidos en sus alcances ya que son útiles para simplificar solamente funciones de Boole expresadas en las formas normalizadas. A pesar de que ello es una desventaja de los métodos, no es muy crítica, ya que la mayoría de aplicaciones buscan, más la forma normalizada, que cualquier otra forma. Se ha visto de la Figura 3-15 que la ejecución con compuertas, de expresiones en la forma normalizada, consiste a lo sumo en dos niveles de compuertas. Las expresiones que no están en la forma normalizada se ejecutan con más de dos niveles. Humphrey (5) muestra una extensión del método del mapa que produce expresiones simplificadas de multiniveles.

Se debe reconocer que la secuencia del código reflejado escogido para los mapas no es única. Es posible dibujar un mapa y asignar una secuencia binaria de código reflejado a las filas y columnas diferente a la secuencia que se ha venido empleando. Siempre y cuando la secuencia binaria escogida produzca el cambio de un solo bit entre cuadrados adyacentes, se producirá un mapa útil y válido.

Dos versiones alternas de mapas de tres variables que a menudo se encuentran en la literatura de lógica digital se muestran en la Figura 3-29. Los números de los términos mínimos se escriben en cada cuadrado para referencias. En (a), la asignación de las variables a las filas y columnas es diferente de la que se usa en este libro. En (b) se ha rotado el mapa a

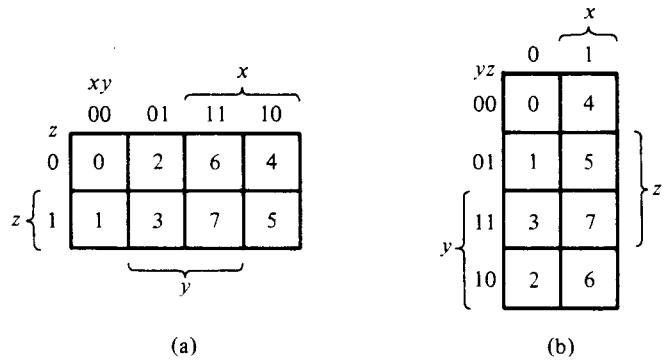


Figura 3-29 Variaciones del mapa de tres variables

la posición vertical. La asignación del número del término mínimo en todos los mapas permanece en el orden xyz. Por ejemplo, el cuadrado del término mínimo 6 se encuentra asignando a las variables ordenadas el número binario $xyz = 110$. El cuadrado para este término mínimo se encuentra en (a) de la columna marcada $xy = 11$ y la fila $z = 0$. El correspondiente cuadrado en (b) pertenece a la columna marcada con $x = 1$ y a la fila con $yz = 10$. El proceso de simplificación con estos mapas es exactamente el mismo que el descrito en este capítulo excepto por supuesto por las variaciones de términos mínimos y la asignación de variables.

Otras dos versiones del mapa de cuatro variables se muestra en la Figura 3-30. El mapa en (a) es muy popular y se usa muy a menudo en la literatura sobre tales temas. De nuevo la diferencia es muy pequeña y se manifiesta por el solo intercambio de la asignación de la variable de filas a columnas y viceversa. El mapa en (b) es el diagrama original de Veitch (1), el cual Karnaugh (2) modificó al mostrado en la Figura (a). Los procesos de simplificación no cambian cuando se usan estos mapas en vez de los usados en este libro. Hay también variaciones de los mapas de cinco o seis variables. De todas maneras, cualquier mapa que parezca diferente al usado en este libro o que se llame de manera diferente, debe reconocer-

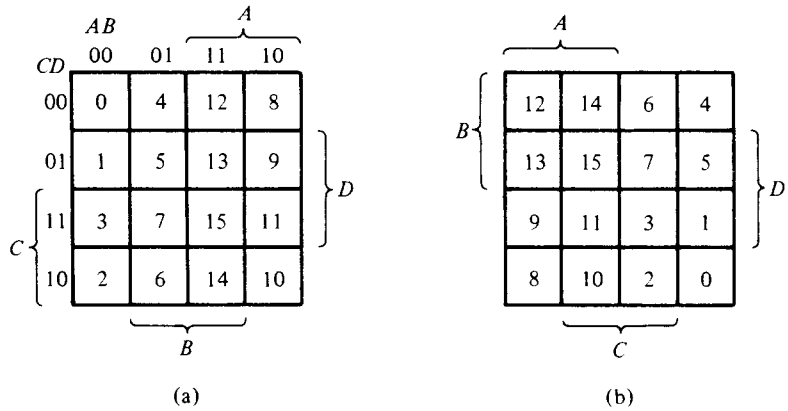


Figura 3-30 Variaciones del mapa de cuatro variables

se simplemente como una variación de la asignación de términos mínimos a los cuadrados del mapa.

Como es evidente de los Ejemplos 3-13 y 3-14, el método del tabulado tiene el inconveniente que ocurren errores inevitables al tratar de comparar los números por medio de listas largas. El método del mapa podría ser preferible, pero para más de cinco variables no se puede estar seguro que se ha encontrado la mejor expresión simplificada. La ventaja real del método del tabulado está en el hecho de que consiste en procedimientos paso a paso que garantizan la respuesta. Es más, este procedimiento formal es adecuado para mecanización por computador.

Se ha establecido en la Sección 3-9 que el método de tabulado siempre comienza con la lista de términos mínimos de la función. Si la función no está en esta forma, debe convertirse a ella. En la mayoría de las aplicaciones, la función que va a ser simplificada proviene de una tabla de verdad, de la cual se puede obtener la lista de términos mínimos. De otra manera, la conversión de términos mínimos agrega un trabajo considerable de manipulación al problema. Sin embargo, existe una extensión del método del tabulado para encontrar los primeros implicados de expresiones algebraicas de suma de productos. Ver por ejemplo McCluskey (7).

En este capítulo se ha considerado la simplificación de funciones con muchas variables de entrada y una sola variable de salida. Sin embargo algunos circuitos digitales tienen más de una salida. Tales circuitos se describen mediante un conjunto de funciones de Boole, una para cada variable de salida. Un circuito con múltiples salidas puede algunas veces tener términos comunes entre las diferentes funciones que pueden ser utilizadas para formar compuertas comunes durante la ejecución. Esto dará como resultado una ulterior simplificación que no se ha considerado cuando cada función se simplifica separadamente. Existe una extensión del método del tabulado para los circuitos de salidas múltiples (6, 7). Sin embargo, este método es muy especializado y bastante tedioso para manipuleo humano. Tiene importancia práctica solamente si se le ofrece al usuario un programa de computador basado en este método.

REFERENCIAS

1. Veitch, E. W., "A Chart Method for Simplifying Truth Functions". *Proc. of the ACM (mayo* 1952), 127-33.
2. Karnaugh, M., "A Map Method for Synthesis of Combinational Logic Circuits". *Trans. AIEE, Comm. and Electronics,* Vol. 72, Parte I (noviembre 1953), 593-99.
3. Quine, W. V., "The Problem of Simplifying Truth Functions". *Am. Math. Monthly,* Vol. 59, No. 8 (octubre 1952), 521-31.
4. McCluskey, E. J., Jr., "Minimization of Boolean Functions". *Bell System Tech. J.,* Vol. 35, No. 6 (noviembre 1956), 1417-44.
5. Humphrey, W. S., Jr., *Switching Circuits with Computer Applications.* Nueva York: McGraw-Hill Book Co., 1958, Capítulo 4.
6. Hill, F. J., y G. R. Peterson, *Introduction to Switching Theory and Logical Design,* 2a. ed. Nueva York: John Wiley & Sons, Inc., 1974, Capítulos 6 y 7.

7. McCluskey, E. J., Jr., *Introduction to the Theory of Switching Circuits*. Nueva York: McGraw-Hill Book Co., 1965, Capítulo 4.

8. Kohavi, Z., *Switching and Finite Automata Theory*. Nueva York: McGraw-Hill Book Co., 1970.

9. Nagle, H. T. Jr., B. D. Carrol, y J. D. Irwin, *An Introduction to Computer Logic*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

PROBLEMAS

3-1. Obtenga las expresiones simplificadas en suma de productos de las siguientes funciones de Boole:

(a) $F(x, y, z) = \Sigma(2, 3, 6, 7)$

(b) $F(A, B, C, D) = \Sigma(7, 13, 14, 15)$

(c) $F(A, B, C, D) = \Sigma(4, 6, 7, 15)$

(d) $F(w, x, y, z) = \Sigma(2, 3, 12, 13, 14, 15)$

3-2. Obtenga las expresiones simplificadas en suma de productos de las siguientes funciones de Boole:

(a) $xy + x'y'z' + x'yz'$

(b) $A'B + BC' + B'C'$

(c) $a'b' + bc + a'bc'$

(d) $xy'z + xyz' + x'yz + xyz$

3-3. Obtenga las expresiones simplificadas en suma de productos de las siguientes funciones de Boole:

(a) $D(A' + B) + B'(C + AD)$

(b) $ABD + A'C'D' + A'B + A'CD' + AB'D'$

(c) $k'lm' + k'm'n + klm'n' + lmn'$

(d) $A'B'C'D' + AC'D' + B'CD' + A'BCD + BC'D$

(e) $x'z + w'xy' + w(x'y + xy')$

3-4. Obtenga las expresiones simplificadas en suma de productos de las siguientes funciones de Boole:

(a) $F(A, B, C, D, E) = \Sigma(0, 1, 4, 5, 16, 17, 21, 25, 29)$

(b) $BDE + B'C'D + CDE + A'B'CE + A'B'C + B'C'D'E'$

(c) $A'B'CE' + A'B'C'D' + B'D'E' + B'CD' + CDE' + BDE'$

3-5. Dada la tabla de verdad:

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Exprese F_1 y F_2 en producto de términos máximos.
(b) Obtenga las funciones simplificadas en suma de productos.
(c) Obtenga las funciones simplificadas en producto de sumas.

3-6. Obtenga las expresiones simplificadas en producto de sumas:
(a) $F(x, y, z) = \Pi(0, 1, 4, 5)$
(b) $F(A, B, C, D) = \Pi(0, 1, 2, 3, 4, 10, 11)$
(c) $F(w, x, y, z) = \Pi(1, 3, 5, 7, 13, 15)$

3-7. Obtenga las expresiones simplificadas en (1) suma de productos y (2) producto de sumas.
(a) $x'z' + y'z' + yz + xyz$
(b) $(A + B' + D)(A' + B + D)(C + D)(C' + D')$
(c) $(A' + B' + D')(A + B' + C')(A' + B + D')(B + C' + D')$
(d) $(A' + B' + D)(A' + D')(A + B + D')(A + B' + C + D)$
(e) $w'yz' + vw'z' + vw'x + v'wz + v'w'y'z'$

3-8. Dibuje la ejecución con compuertas de las funciones de Boole simplificadas, obtenidas en el Problema 3-7 usando las compuertas AND y OR.

3-9. Simplifique cada una de las siguientes funciones y ejecútelas con compuertas NAND. Dar dos alternativas.
(a) $F_1 = AC' + ACE + ACE' + A'CD' + A'D'E'$
(b) $F_2 = (B' + D')(A' + C' + D)(A + B' + C' + D)(A' + B + C' + D')$

3-10. Repita el Problema 3-9 para ejecuciones con NOR.

3-11. Ejecute las funciones siguientes con compuertas NAND. Asuma que se cuenta con entradas normales y complementadas.
(a) $BD + BCD + AB'C'D' + A'B'CD'$ con no más de seis compuertas, cada una con tres entradas.
(b) $(AB + A'B')(CD' + C'D)$ con dos compuertas de dos entradas.

3-12. Ejecute las siguientes funciones con compuertas NOR. Asuma que se cuenta con las entradas normal y complementada.
(a) $AB' + C'D' + A'CD' + DC'(AB + A'B') + DB(AC' + A'C)$
(b) $AB'CD' + A'BCD' + AB'C'D + A'BC'D$

3-13. Haga una lista de las formas degeneradas de dos niveles y demuestre que se reducen a una sola operación. Explique cómo las formas degeneradas de dos niveles pueden ser usadas para aumentar el *fan-out* de las compuertas.

3-14. Ejecute las funciones del Problema 3-9 con las siguientes formas de dos niveles: NOR-OR, NAND-AND, OR-NAND y AND-NOR.

3-15. Simplifique las funciones de Boole F en suma de productos usando las condiciones de no importa d:
(a) $F = y' + x'z'$
$d = yz + xy$
(b) $F = B'C'D' + BCD' + ABCD'$
$d = B'CD' + A'BC'D$

3-16. Simplifique la función de Boole F usando las condiciones de no importa d en (1) suma de productos y (2) producto de sumas:

(a) $F = A'B'C' + A'CD + A'BC$
$d = A'BC'D + ACD + AB'D'$

(b) $F = w'(x'y + x'y' + xyz) + x'z'(y + w)$
$d = w'x(y'z + yz') + wyz$

(c) $F = ACE + A'CD'E' + A'C'DE$
$d = DE' + A'D'E + AD'E'$

(d) $F = B'DE' + A'BE + B'C'E' + A'BC'D'$
$d = BDE' + CD'E'$

3-17. Ejecute las siguientes funciones usando las condiciones de no importa. Asuma que se cuenta con las entradas normales y sus complementos.

(a) $F = A'B'C' + AB'D + A'B'CD'$ con dos compuertas NOR a lo sumo.
$d = ABC + AB'D'$

(b) $F = (A + D)(A' + B)(A' + C')$ con tres compuertas NAND a lo sumo.

(c) $F = B'D + B'C + ABCD$ con compuertas NAND.
$d = A'BD + AB'C'D'$

3-18. Ejecute las siguientes funciones en compuertas NAND y NOR. Use solamente cuatro compuertas. Solamente se cuenta con las entradas normales.

$$F = w'xz + w'yz + x'yz' + wxy'z$$
$$d = wyz$$

3-19. La siguiente expresión de Boole:

$$BE + B'DE'$$

es la versión simplificada de la función:

$$A'BE + BCDE + BC'D'E + A'B'DE' + B'C'DE'$$

¿Hay condiciones de no importa? Si es así, ¿cuáles son ellas?

3-20. Dé tres maneras posibles de expresar las funciones:

$$F = A'B'D' + AB'CD' + A'BD + ABC'D$$

con ocho o menos literales.

3-21. Con el uso de mapas, encuentre la forma más simple en suma de productos de la función $F = fg$, donde f y g estén dados por:

$$f = wxy' + y'z + w'yz' + x'yz'$$
$$g = (w + x + y' + z')(x' + y' + z)(w' + y + z')$$

Sugerencia: Ver el Problema 2-8(b).

3-22. Simplifique la función de Boole del Problema 3-2(a) usando el mapa definido en la Figura 3-29(a). Repita el ejercicio con el mapa de la Figura 3-29(b).

3-23. Simplifique la función de Boole del Problema 3-3(a) usando el mapa definido en la Figura 3-30(a). Repita con el mapa de la Figura 3-30(b).

3-24. Simplifique las siguientes funciones de Boole por medio del método del tabulado.

(a) $F(A, B, C, D, E, F, G) = \Sigma(20, 28, 52, 60)$

(b) $F(A, B, C, D, E, F, G) = \Sigma(20, 28, 38, 39, 52, 60, 102, 103, 127)$

(c) $F(A, B, C, D, E, F) = \Sigma(6, 9, 13, 18, 19, 25, 27, 29, 41, 45, 57, 61)$

3-25. Repita el Problema 3-6 mediante el uso del método del tabulado.

3-26. Repita el Problema 3-16(c) y (d) usando el método del tabulado.

Lógica combinacional

4-1 INTRODUCCION

Los circuitos lógicos para los sistemas digitales pueden ser combinacionales o secuenciales. Un circuito combinacional consiste en compuertas lógicas cuyas salidas se determinan directamente en cualquier momento de la combinación presente de entradas sin tener en cuenta las entradas anteriores. Un circuito combinacional realiza una operación de procesamiento de información específica completamente lógica por medio de un conjunto de funciones de Boole. Los circuitos secuenciales usan elementos de memoria (celdas binarias), además de compuertas lógicas. Sus salidas son una función de las entradas y del estado de los elementos de la memoria. El estado de los elementos de la memoria, a su vez es una función de las entradas previas. Como consecuencia, las salidas de un circuito secuencial dependen no solamente de las entradas presentes, sino también de las entradas pasadas, y el comportamiento del circuito debe especificarse por una secuencia de tiempos de las entradas y estados internos. Los circuitos secuenciales se discuten en el Capítulo 6.

En el Capítulo 1 se aprendió a reconocer los números y códigos binarios que representan las cantidades discretas de información. Estas variables binarias se representan por medio de voltajes eléctricos o por cualquier otra señal. Las señales pueden ser manipuladas por compuertas lógicas digitales con el fin de ejecutar las funciones deseadas. En el Capítulo 2 se introdujo el álgebra de Boole como vehículo para expresar algebraicamente funciones lógicas. En el Capítulo 3 se aprendió a simplificar las funciones de Boole para lograr ejecuciones con compuertas de tipo económico. El propósito de este capítulo es el de usar los conocimientos adquiridos en los Capítulos anteriores y el de formular varios diseños sistemáticos y procedimientos de análisis de los circuitos combinacionales. La solución de algunos ejemplos típicos dará una recopilación útil de funciones elementales importantes para la comprensión de computadores digitales y sistemas.

Un circuito combinacional consiste en variables de entrada, compuertas lógicas y variables de salida. Las compuertas lógicas aceptan señales

Figura 4-1 Diagrama de bloque de un circuito combinacional

en las entradas y generan señales en las salidas. Este proceso trasforma información binaria de datos de entrada dados a datos de salida requeridos. Obviamente, los datos de salida y de entrada se representan por medio de señales binarias, es decir, existen dos valores posibles, uno representado *lógica 1* y el otro representado *lógica 0*. En la Figura 4-1 se muestra un diagrama de bloque de un circuito combinacional. Las n variables binarias de entrada vienen de una fuente externa, las m variables de salida van a un destino externo. En muchas aplicaciones la fuente y el destino son registros acumuladores (Sección 1-7) localizados en la vecindad de un circuito combinacional o en algún componente remoto externo. Por definición, un registro externo no debe influenciar el comportamiento de un circuito combinacional ya que si lo hace el sistema total se convierte en un circuito secuencial.

Para n variables de entrada, hay 2^n combinaciones posibles de valores de entrada binaria. Para cada combinación de entrada posible hay una y sólo una combinación de salida posible. Un circuito combinacional puede describirse por m funciones de Boole, una para cada variable de salida. Cada función de salida se expresa en términos de n variables de entrada.

Cada variable de entrada a un circuito combinacional puede tener una o dos conexiones. Cuando se cuenta solamente con una conexión, se puede representar la variable en la forma normal (no tildada) o en la forma de complemento (tildada). Como una variable en una expresión de Boole puede aparecer tildada y no tildada es necesario suministrar un inversor para cada literal que no se obtenga en el terminal de entrada. Por otra parte, una variable de entrada puede aparecer en dos terminales suministrando las formas normales y de complemento a la entrada del circuito. Si este es el caso, no es necesario incluir los inversores a las entradas. El tipo de celdas binarias usadas en la mayoría de los sistemas digitales son circuitos flip-flops (Capítulo 6) que tienen salidas para los valores normales y complementados de la variable binaria acumulada. En el trabajo subsiguiente, se asume que cada variable de entrada aparece en dos terminales, suministrando simultáneamente los valores normales y de complemento. Se debe tener en cuenta que un circuito inversor puede producir el complemento de la variable si se cuenta con un solo terminal.

4-2 PROCEDIMIENTO DE DISEÑO

El diseño de circuitos combinacionales comienza desde el enunciado del problema y termina con el diagrama de circuito lógico, o con un conjunto de funciones de Boole de los cuales se puede obtener el diagrama lógico fácilmente. El procedimiento cubre los siguientes pasos:

1. Se enuncia el problema.
2. Se determina el número requerido de variables de entrada y el número requerido de variables de salida.
3. Se le asignan letras a las variables de entrada y salida.
4. Se deduce la tabla de verdad que define las relaciones entre las entradas y las salidas.
5. Se obtiene la función de Boole simplificada para cada salida.
6. Se dibuja el diagrama lógico.

Una tabla de verdad para circuitos combinacionales consiste en columnas de entrada y columnas de salida. Los unos y ceros en las columnas de entrada se obtienen de las 2^n combinaciones binarias disponibles para n variables de entrada. Los valores binarios para las salidas se determinan después de un examen del problema enunciado. Una salida puede ser igual a 0 ó 1 para cada combinación válida de entrada. Sin embargo, las especificaciones podrían indicar que algunas combinaciones de entrada no ocurrirán. Estas combinaciones se convertirán en condiciones de no importa.

Las funciones de salida especificadas en la tabla de verdad darán la definición exacta del circuito combinacional. Es importante que las especificaciones enunciadas se interpreten correctamente en la tabla de verdad. Algunas veces el diseñador debe usar su intuición y experiencia para llegar a la interpretación correcta. Las especificaciones enunciadas son rara vez completas y exactas. Cualquier interpretación errónea que produzca una tabla de verdad incorrecta dará como resultado un circuito combinacional que no cubra las necesidades establecidas.

Las funciones de Boole de salida de una tabla de verdad se simplifican por cualquier método disponible, tal como manipulación algebraica, el método del mapa o el procedimiento del tabulado. Normalmente habrá una variedad de expresiones simplificadas entre los cuales se puede escoger. Sin embargo, en una aplicación particular, ciertas restricciones, limitaciones y criterios vienen como guía en el proceso de selección de una expresión algebraica particular. Un método práctico de diseño tendrá que considerar tales condiciones obligatorias como (1) número mínimo de compuertas, (2) número mínimo de entradas a una compuerta, (3) tiempo de propagación mínima de una señal a través del circuito, (4) número mínimo de interconexiones y (5) limitaciones de la capacidad de accionamiento de cada compuerta. Como todos estos criterios no pueden satisfacerse simultáneamente y como la importancia de las condiciones obligatorias se dictan para la aplicación particular, es difícil hacer una afirmación general en lo que respecta a una simplificación aceptable. En la mayoría de los casos la simplificación comienza por lograr un objetivo elemental, tal como producir una función de Boole simplificada en la forma normalizada y de allí proceder a lograr los otros criterios de comportamiento.

En la práctica, los diseñadores tienden a ir de las funciones de Boole a una lista de terminales que muestran las interconexiones entre varias

compuertas lógicas normalizadas. En este caso el diseño no debe ir más allá de las funciones de Boole simplificadas de salida. Sin embargo, el diagrama lógico es útil para visualizar la ejecución de las expresiones con compuertas.

4-3 SUMADORES

Las compuertas digitales hacen una variedad de tareas de procesamiento de información. Entre las funciones básicas encontradas están las diferentes operaciones aritméticas. La operación aritmética más básica es sin duda la suma de dos dígitos binarios. Esta simple adición consiste en cuatro operaciones elementales posibles así: $0+0=0$, $0+1=1$, $1+0=1$ y $1+1=10$. Las primeras tres operaciones producen una suma cuya longitud es en un dígito, pero, en el caso en que ambos sumandos sean iguales a 1 la suma binaria consiste en dos dígitos. El bit más significativo de ese resultado se llama *bit de arrastre (acarreo)*. Cuando los números de los sumandos contienen más dígitos significativos, el bit de arrastre que se obtiene de la suma de dos bits se agrega al siguiente par de bits significativos de mayor orden. Un circuito combinacional que realiza la suma de dos bits se llama *sumador medio*. Aquel que realiza la suma de tres bits (dos bits significativos más el bit de arrastre) es un *sumador completo*. El nombre del primero se deriva del hecho de que se usan dos sumadores medios para hacer un sumador completo. Los dos circuitos sumadores no son los primeros circuitos combinacionales que se van a diseñar.

Sumador medio

De la explicación verbal del sumador medio se encuentra que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits de los sumandos, las variables de salida producen la suma y el bit de arrastre. Es necesario especificar dos variables de salida porque el resultado puede consistir de dos dígitos binarios. Se asignan arbitrariamente los símbolos x y y a las dos entradas, S (para la suma) y C (para el bit de arrastre) para las salidas.

Una vez que se haya establecido el número y los nombres de las variables de entrada y salida se está listo para formular la tabla de verdad para identificar exactamente la función del sumador medio. Esta tabla de verdad se muestra a continuación:

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

El bit de arrastre es 0 a no ser que ambas entradas sean 1. La salida S representa el bit menos significativo de la suma.

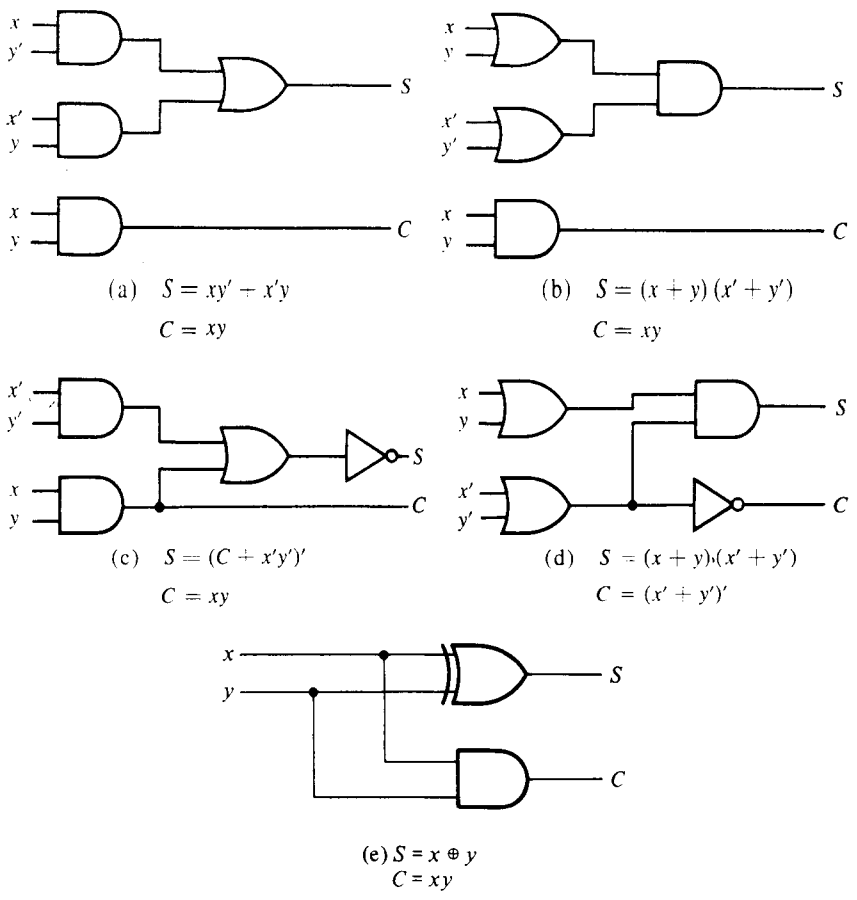


Figura 4-2 Varias configuraciones del sumador medio

Las funciones de Boole simplificadas para las dos salidas pueden obtenerse directamente de una tabla de verdad. Las expresiones simplificadas en suma de productos son:

$$S = x'y + xy'$$
$$C = xy$$

El diagrama lógico para esta configuración se muestra en la Figura 4-2(a) de la misma manera que otras cuatro formas para hacer un sumador medio. Todas ellas logran el mismo resultado en cuanto al comportamiento de entrada-salida. Ellas muestran la flexibilidad disponible para el diseñador cuando se configura una función lógica combinacional simple, tal como ésta.

La Figura 4-2(a), como se ha enunciado antes, es la configuración del sumador medio en suma de productos. La Figura 4-2(b) muestra la configuración en producto de sumas:

$$S = (x + y)(x' + y')$$
$$C = xy$$

Para obtener la configuración de la Figura 4-2(c), se nota que S es la OR-exclusiva de x y y. El complemento de S es el equivalente de x y y (Sección 2-6):

$$S' = xy + x'y'$$

pero como $C = xy$ se obtiene:

$$S = (C + x'y')'$$

En la Figura 4-2(d) se usa la configuración del producto de sumas con C derivado como sigue:

$$C = xy = (x' + y')'$$

El sumador medio puede ser configurado con una OR-exclusiva y una compuerta AND de la manera mostrada en la Figura 4-2(e). Esta forma se usa más tarde para demostrar que se necesitan dos sumadores medios para construir un circuito sumador completo.

Sumador completo

Un sumador completo es un circuito combinacional que forma la suma aritmética de tres bits de entrada. Este consiste en tres entradas y dos salidas. Dos de las variables de entrada denotadas por x y y representan los dos bits significativos que se agregan. La tercera entrada z representa el bit de arrastre de la posición previa menos significativa. Se necesitan dos salidas porque la suma aritmética de tres dígitos binarios varía en valor de 0 a 3 y los binarios 2 ó 3 necesitan dos dígitos. Las dos salidas se designan por los símbolos S para la suma y C para el bit de arrastre. La variable binaria S da el valor de la suma del bit menos significativo. La variable binaria C da el bit de arrastre de salida. La tabla de verdad del sumador completo es como sigue a continuación:

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Las ocho filas debajo de las variables de entrada designan todas las combinaciones posibles de unos y ceros que pueden tener esas variables. Los unos y ceros de las variables de salida se determinan por la suma aritmética de los bits de entrada. Cuando todos los bits de entrada sean ceros, la salida es cero. La salida S es igual a 1 cuando solamente una entrada

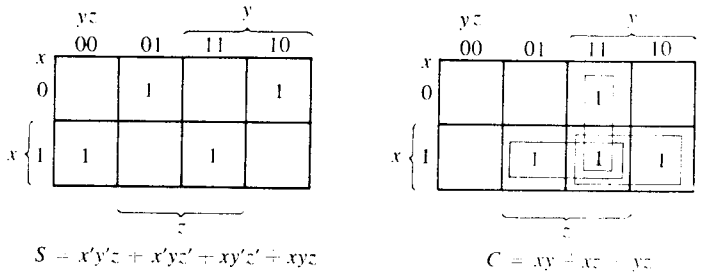


Figura 4-3 Mapas de un sumador completo

es igual a 1 ó cuando todas las tres entradas sean iguales a uno. La salida C tiene un bit de arrastre de 1 si dos de las tres entradas son iguales a 1.

Los bits de entrada y salida de los circuitos combinacionales tienen diferentes interpretaciones en los diferentes estados del problema. Físicamente, las señales binarias de los terminales de entrada se consideran dígitos binarios agregados aritméticamente para formar una suma de dos dígitos en los terminales de salida. Por otra parte, los mismos valores binarios se consideran variables de las funciones de Boole cuando se expresan en la tabla de verdad o cuando se ejecutan los circuitos con compuertas lógicas. Es importante tener en cuenta que se dan dos interpretaciones diferentes a los valores de los bits encontrados en este circuito.

La relación lógica de entrada-salida del circuito del sumador completo puede ser expresada con dos funciones de Boole, una para cada variable de salida. Cada función de Boole de salida requiere un mapa único para su simplificación. Cada mapa debe tener ocho cuadrados ya que cada salida es una función de las tres variables de entrada. Los mapas de la Figura 4-3 se usan para simplificar las dos funciones de salida. Los unos en los cuadrados de los mapas para S y C se determinan directamente de la tabla de verdad. Los cuadrados con unos para la salida S, no combinan en cuadrados adyacentes, para dar una expresión simplificada en suma de productos. La salida C puede simplificarse a una expresión de 6 literales. El diagrama lógico para el sumador completo ejecutado en suma de productos se muestra en la Figura 4-4. Esta configuración usa las siguientes expresiones de Boole.

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

Se pueden desarrollar otras configuraciones para el sumador completo. La ejecución del producto de sumas requiere el mismo número de compuertas que la configuración de la Figura 4-4, con el grupo de compuertas AND y OR intercambiadas. Un sumador completo puede configurarse con dos sumadores medios y una compuerta OR, como se muestra en la Figura 4-5. La salida S del segundo sumador medio es la aplicación de una OR-exclusiva de z y la salida del primer sumador medio dando:

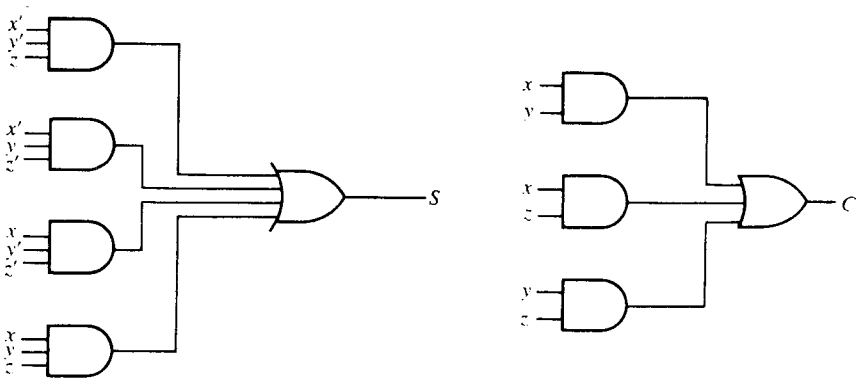


Figura 4-4 Configuración de un sumador completo en suma de productos

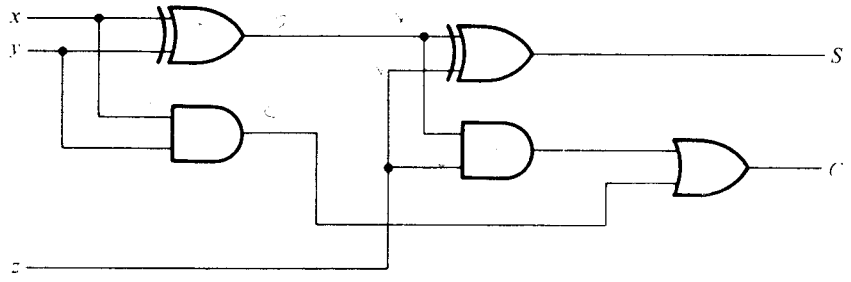


Figura 4-5 Configuración de un sumador completo con dos sumadores medios y una compuerta OR

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

y el bit de arrastre de salida será:

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

4-4 SUSTRACTORES

La sustracción de dos números binarios pueden lograrse tomando el complemento del sustraendo para agregarlo al minuendo (Sección 1-5). Mediante este método, la operación de sustracción se convierte en operación de suma que necesita *sumadores completos* para su ejecución en una máquina. Es posible ejecutar la sustracción con circuitos lógicos de una manera directa como se hace con lápiz y papel. Mediante este método, cada bit de sustraendo del número se resta de su correspondiente bit significativo del minuendo para formar el bit de la diferencia. Si el bit del minuendo es menor que el bit del sustraendo, se presta un 1 de la siguiente posición significativa. El hecho de que se ha prestado un 1 debe llevarse

al siguiente par de bits mayores por medio de las señales binarias que vienen (salida) de un estado dado y van al (entrada) siguiente estado mayor. De la misma manera que hay sumadores medios y completos. Hay sustractores medios y completos.

Sustractor medio

Un sustractor medio es un circuito combinacional que resta dos bits y produce su diferencia. Este también tiene una salida que especifica si se ha prestado un 1. Se designa el bit del minuendo con x y el bit del sustraendo con y. Para realizar $x-y$ se debe constatar las magnitudes relativas de x y y. Si $x \geq y$, tendremos tres posibilidades: $0-0=0$, $1-0=1$ y $1-1=0$. El resultado se llama el *bit de diferencia*. Si $x<y$ se tiene $0-1$, y se hace necesario prestar un 1 del siguiente estado mayor. El 1 prestado del estado siguiente mayor agrega 2 al bit del minuendo, de la misma forma que en el sistema decimal un número prestado agrega 10 al dígito del minuendo. Con el minuendo igual a 2 la diferencia se convierte en $2-1=1$. El sustractor medio necesita dos salidas. Una salida genera la diferencia y se designa mediante el símbolo D. La segunda salida designada como B (B viene de Borrow), genera la señal binaria que informa al siguiente estado que se ha prestado un uno. La tabla de verdad para las relaciones de entrada-salida de un sustractor medio se puede derivar de la siguiente manera:

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

La salida prestada B es 0 siempre y cuando $x \geq y$. Será 1 para $x=0$ y $y=1$. La salida D es el resultado de la operación aritmética $2B+x-y$.

Las funciones de Boole para las dos salidas del sustractor medio se derivan directamente de la tabla de verdad:

$$D = x'y + xy'$$
$$B = x'y$$

Es interesante notar que la lógica para D es exactamente la misma que la lógica para la salida S del sumador medio.

Sustractor completo

Un sustractor completo es un circuito combinacional que realiza una resta entre dos bits, tomando en consideración que se ha prestado un 1 de un estado menos significativo. Este circuito tiene tres entradas y dos salidas. Las tres entradas, x, y y z denotan el minuendo, el sustraendo y el bit de arrastre o bit prestado respectivamente. Las dos salidas, D y B, represen-

tan la diferencia y la salida del bit prestado respectivamente. La tabla de verdad para este circuito es la siguiente:

x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Las ocho filas debajo de las variables de entrada designan todas las combinaciones posibles de unos y ceros que pueden adoptar las variables binarias. Los unos y ceros para las variables de salida se determinan por la resta de $x-y-z$. Las combinaciones que tienen entrada prestada $z=0$ se reducen a las mismas cuatro condiciones del sumador medio. Para $x=0$, $y=0$ y $z=1$ es necesario prestar un 1 del siguiente estado, lo cual hace $B=1$ y agregar 2 a x. Ya que $2-0-1=1$, $D=1$. Para $x=0$ y $yz=11$, es necesario prestar de nuevo haciendo $B=1$ y $x=2$. Ya que $2-1-1=0$. $D=0$. Para $x=1$ y $yz=01$, se tiene $x-y-z=0$ lo cual hace $B=0$ y $D=0$. Finalmente para $x=1$ y $y=1$, $z=1$ se tiene que prestar 1, haciendo $B=1$ y $x=3$ para $3-1-1=1$ haciendo $D=1$.

Las funciones de Boole simplificadas para las dos salidas del sustractor completo se derivan de los mapas de la Figura 4-6. Las funciones simplificadas en suma de productos serán:

$$D = x'y'z + x'yz' + xy'z' + xyz$$
$$B = x'y + x'z + yz$$

De nuevo se nota que la función lógica para la salida D en un sustractor completo es exactamente la misma que la salida S en el sumador completo. Sin embargo, la salida B se parece a la función C en el sumador completo, excepto que la variable de entrada x se complementa. Debido a estas similitudes, es posible convertir un sumador completo a un sustractor

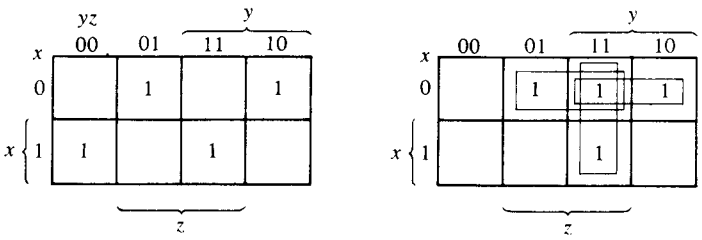


Figura 4-6 Mapas para un sumador completo

completo simplemente complementando la entrada x antes de su aplicación a las compuertas que forman el bit de arrastre de salida.

4-5 CONVERSION ENTRE CODIGOS

La disponibilidad de una gran variedad de códigos para los mismos elementos discretos de información da como resultado el uso de códigos diferentes para diferentes sistemas digitales. Es necesario algunas veces usar la salida de un sistema como entrada de otro. Un circuito de conversión debe colocarse entre los dos sistemas, si cada uno usa diferentes códigos para la misma información. De esta forma un conversor de código es un circuito que hace compatibles dos sistemas a pesar de que ambos tengan diferente código binario.

Para convertir el código binario A al código binario B, las líneas de entrada deben dar una combinación de bits de los elementos, tal como se especifica por el código A y las líneas de salida deben generar la correspondiente combinación de bits del código B. Un circuito combinacional realiza esta trasformación por medio de compuertas lógicas. El procedimiento de diseño de los conversores de código se ilustra mediante un ejemplo específico de conversión de BDC a código de exceso 3.

Las combinaciones de bits del BDC y el exceso 3 se listan en la Tabla 1-2 (Sección 1-6). Como cada código usa cuatro bits para representar un dígito decimal, debe haber cuatro variables de entrada y cuatro variables de salida. Es conveniente designar las cuatro variables binarias de entrada mediante los símbolos A, B, C y D y las cuatro variables de salida con w, x, y, y z. La tabla de verdad que relaciona las variables de entrada y salida se muestran en la Tabla 4-1. Las combinaciones de bits para las entradas y sus correspondientes salidas se obtienen directamente de la Tabla 1-2. Se nota que cuatro variables binarias pueden tener 16 combinaciones

Tabla 4-1 Tabla de verdad para el ejemplo de conversión de código

Entrada BDC				Salida código exceso 3			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

de bits de las cuales se listan 10 en la tabla de verdad. Las seis combinaciones de bits no listadas para las variables *entrada* son las combinaciones de no importa. Como ellas nunca ocurren, se tiene la libertad de asignar un 1 ó un 0, a las variables de salida, de acuerdo a la que dé un circuito más simple.

Los mapas de la Figura 4-7 se dibujan para obtener una función de Boole simplificada para cada salida. Cada uno de los cuatro mapas de la Figura 4-7 representa una de las cuatro salidas de este circuito como función de las cuatro variables de entrada. Los unos marcados dentro de los cuadrados, se obtienen de dos términos mínimos que hacen que la salida sea igual a 1. Los unos se obtienen de la tabla de verdad observando las columnas de salida una por una. Por ejemplo, la columna bajo la salida z tiene 5 unos, por tanto, el mapa para z debe tener cinco unos cada uno de los cuales debe ser un cuadrado que corresponde al término mínimo que hace z igual a 1. Las seis combinaciones de no importa se marcan con X. Una posible forma de simplificar las funciones en suma de productos se lista bajo el mapa de cada variable.

Se puede obtener un diagrama lógico de dos niveles directamente de las expresiones de Boole derivadas de los mapas. Hay otras posibilidades para el diagrama lógico que ejecuta este circuito. Las expresiones obteni-

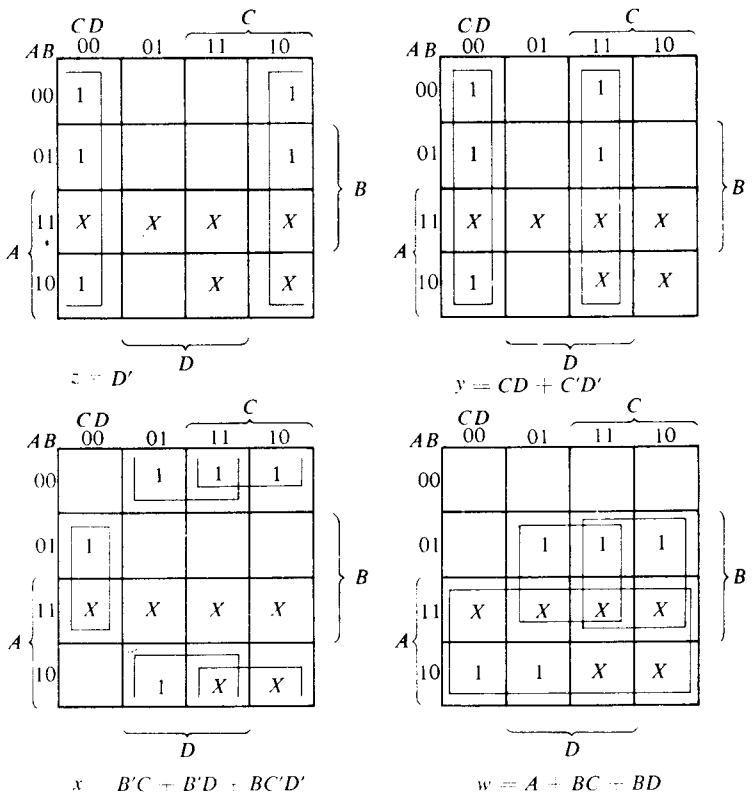


Figura 4-7 Mapas para el conversor de código de BDC exceso 3

das en la Figura 4-7 pueden manipularse algebraicamente con el propósito de usar compuertas comunes para dos o más salidas. Esta manipulación mostrada a continuación, ilustra la flexibilidad obtenida con los sistemas de múltiples salidas cuando se ejecutan con tres o más niveles de compuertas.

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

El diagrama lógico que configura la expresión anterior se muestra en la Figura 4-8. En este se observa que la compuerta OR cuya salida es $C + D$ se ha usado para configurar parcialmente cada una de las tres salidas.

No teniendo en cuenta los inversores de entrada, la ejecución en suma de productos requiere siete compuértas AND y tres compuertas OR. La configuración de la Figura 4-8 requiere cuatro compuertas AND, cuatro compuertas OR y un inversor. Si están disponibles solamente las entradas normales, la primera ejecución requerirá inversores para las variables B, C y D. Mientras que la segunda ejecución requiere inversores para las variables B y D.

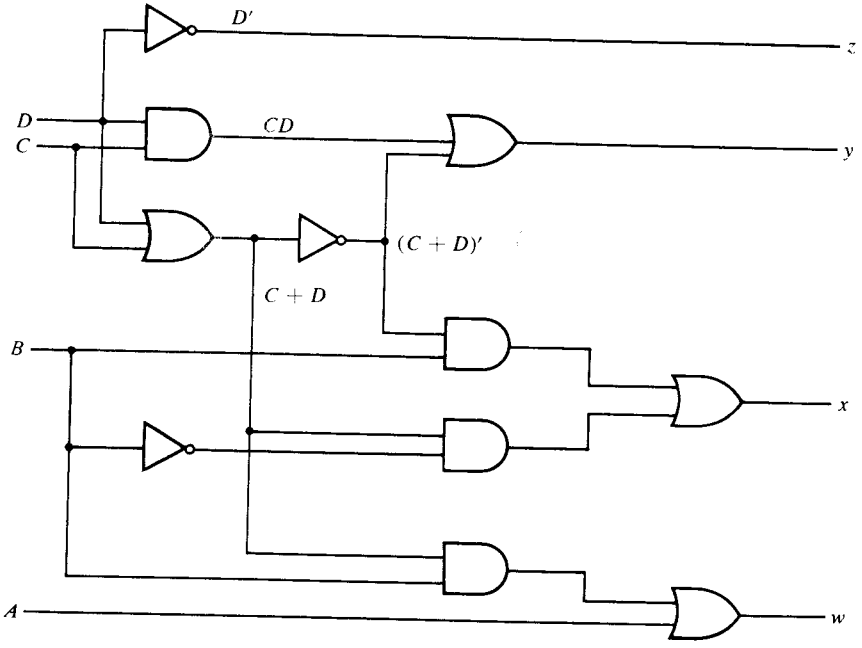


Figura 4-8 Diagrama lógico para el conversor de código BDC a exceso 3

4-6 PROCEDIMIENTO DE ANALISIS

El diseño de los circuitos combinacionales comienza con las especificaciones enunciadas de una función requerida y culmina con un conjunto de funciones de Boole de salida o un diagrama lógico. El *análisis* de un circuito combinacional es de cierta manera el proceso inverso. Este comienza con un diagrama lógico dado y culmina con un conjunto de funciones de Boole, una tabla de verdad o una explicación verbal de la operación del circuito. Si el diagrama lógico que se va a analizar se acompaña del nombre de la función, o una explicación de lo que se asume que logre, entonces el análisis del problema se reduce a la verificación de la función enunciada.

El primer paso en el análisis es asegurarse que el circuito dado sea combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin caminos de realimentación o elementos de memoria. Un camino de realimentación es una conexión de la salida de una compuerta a la entrada de una segunda compuerta que forma parte de la entrada de la primera compuerta. Los caminos de realimentación o elementos de memoria en un circuito digital definen un circuito secuencial y deben ser analizados de acuerdo a los procedimientos esbozados en el Capítulo 6.

Una vez que se verifique el diagrama lógico como circuito combinacional, se puede proceder a obtener las funciones de salida y la tabla de verdad. Si el circuito se acompaña de una explicación verbal de esta función, entonces las funciones de Boole o la tabla de verdad son suficientes para la verificación. Si la función del circuito está bajo investigación, entonces es necesario interpretar la operación del circuito de la tabla de verdad derivada. El éxito de tal investigación se facilita si se tiene experiencia previa y familiaridad con una gran variedad de circuitos digitales. La habilidad de correlacionar una tabla de verdad con una tarea de procesamiento de información es un arte que se adquiere con la experiencia.

Para obtener las funciones de Boole de salida de un diagrama lógico, se procede de la siguiente manera:

1. Señálese con símbolos arbitrarios todas las salidas de las compuertas que son función de las variables de entrada. Obténgase las funciones de Boole para cada compuerta.
2. Márquese con otros símbolos arbitrarios aquellas compuertas que son una función de las variables de entrada y las compuertas marcadas anteriormente. Encuéntrese las funciones de Boole para ellas.
3. Repítase el proceso esbozado en el paso 2 hasta que se obtengan las salidas del circuito.
4. Obténgase las funciones de Boole de salida en términos de las variables de entrada solamente, por sustitución repetida de las funciones definidas anteriormente.

El análisis del circuito combinacional en la Figura 4-9 ilustra el procedimiento propuesto. Se nota que el circuito tiene tres entradas binarias,

A, B y C y dos salidas binarias, F_1 y F_2. Las salidas de las diferentes compuertas se marcan con símbolos intermedios. Las salidas de las compuertas que son funciones de las variables de entrada son solamente F_2, T_1 y T_2. Las funciones de Boole para estas tres salidas son:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

En seguida se consideran las compuertas de salida que son funciones de los símbolos ya definidos:

$$T_3 = F_2' T_1$$
$$F_1 = T_3 + T_2$$

La función de Boole de salida F_2 está ya expresada como una función de las entradas solamente. Para obtener F_1 como función de A, B y C se forman una serie de sustituciones como sigue a continuación:

$$\begin{aligned} F_1 &= T_3 + T_2 = F_2' T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Si se quiere continuar con la investigación y determinar la tarea de información-trasformación lograda por este circuito se puede derivar la tabla de verdad directamente de las funciones de Boole y tratar de reco-

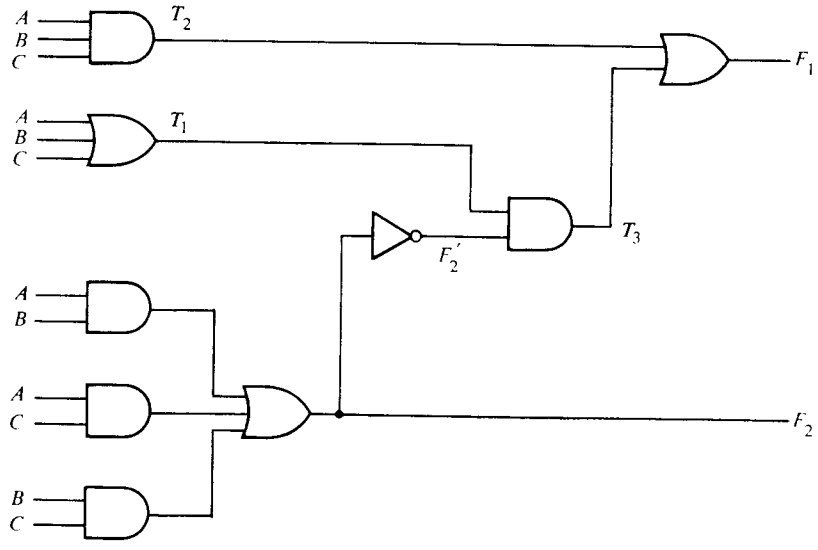


Figura 4-9 Diagrama lógico para el ejemplo de análisis

nocer una operación familiar. Para este ejemplo nótese que el circuito es un sumador completo, con F_1 siendo la suma de salida y F_2 el bit de arrastre de salida. A, B y C son las tres entradas sumadas algebraicamente.

La derivación de la tabla de verdad para el circuito es un proceso directo una vez que se reconozcan las funciones de Boole de salida. Para obtener la tabla de verdad directamente del diagrama lógico sin pasar por las derivaciones de las funciones de Boole, se procede de la siguiente manera:

1. Determínese el número de variables de entrada del circuito. Para n entradas, fórmese las 2^n posibles combinaciones de entrada de unos y ceros listando los números binarios desde 0 hasta $2^n - 1$.
2. Márquese las salidas de las compuertas seleccionadas con símbolos arbitrarios.
3. Obténgase la tabla de verdad para las salidas de aquellas compuertas que son una función de las variables de entrada solamente.
4. Procédase a obtener la tabla de verdad para las salidas de aquellas compuertas que son una función de los valores definidos previamente hasta que se determinen las columnas para todas las salidas.

Este proceso puede ilustrarse usando el circuito de la Figura 4-9. En la Tabla 4-2 se forman las ocho combinaciones posibles para las tres entradas variables. La tabla de verdad para F_2 se determina directamente de los valores de A, B y C con F_2 igual a 1 para cualquier combinación que tiene dos o tres entradas iguales a 1. La tabla de verdad para F_2' es el complemento de F_2. Las tablas de verdad para T_1 y T_2 son las funciones OR y AND de las variables de entrada respectivamente. Los valores para T_3 se derivan de T_1 y F_2': T_3 es igual a 1 cuando T_1 y F_2' son iguales a uno, y a cero de otra manera. Finalmente, F_1 es igual a 1, para aquellas combinaciones en las cuales T_2 o T_3 o ambas sean iguales a 1. Por inspección de las combinaciones de la tabla de verdad para A, B, C, F_1 y F_2 de la Tabla 4-2 se muestra que son idénticas a la tabla de verdad del sumador completo dado en la Sección 4-3 para x, y, z, S y C respectivamente.

Tabla 4-2 Tabla de verdad para el diagrama lógico de la Figura 4-9

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

Considérese ahora un circuito combinacional que tiene combinaciones de entrada de no importa. Cuando se diseña un circuito como este, se marcan las combinaciones de no importa con una X en el mapa y se les asigna un 1 o un 0, según sea lo más conveniente para la simplificación de la función de Boole de salida. Cuando se analiza un circuito con combinaciones de no importa se tiene una situación totalmente diferente. Aunque se asume que las combinaciones de entrada de no importa nunca ocurren, el hecho es que si cualquiera de estas combinaciones se aplica a las entradas (intencionalmente o por error) se tendrá presente una salida binaria. El valor de la salida dependerá de la escogencia de la X durante el diseño. Parte del análisis de tal circuito puede involucrar la determinación de los valores de salida para las combinaciones de entrada de no importa. Como ejemplo, considérese el conversor de código de BDC a código de exceso 3 diseñado en la Sección 4-5. Las salidas obtenidas cuando se aplican las seis combinaciones no usadas del código BDC a las entradas son:

Entradas BDC no usadas				*Salidas*			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
1	0	1	0	1	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	1	1	1	1
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	1
1	1	1	1	1	0	1	0

Estas salidas pueden derivarse por medio del método del análisis de la tabla de verdad esbozado en esta sección. En este caso particular, las salidas pueden obtenerse directamente de los mapas de la Figura 4-7. Por inspección de los mapas, se determina cuando las X en los cuadrados de los términos mínimos correspondientes a cada salida, han sido incluidos como unos o ceros. Por ejemplo, el cuadrado del término mínimo m_{10} (1010) se ha incluido con los unos para dar salidas w, x y z pero no para y. Por tanto, las salidas para m_{10} son $wxyz = 1101$ tal como están listadas en la tabla anterior. Se nota que las primeras tres salidas en la tabla no tienen significado en el código de exceso 3 y por lo menos tres salidas corresponden al decimal 5, 6 y 7 respectivamente. Esta coincidencia es totalmente una función de la escogencia de X durante el diseño.

4-7 CIRCUITOS NAND DE MULTINIVEL

Los circuitos combinacionales se construyen más frecuentemente con compuertas NAND y NOR en vez de compuertas AND y OR. Las compuertas NAND y NOR son más comunes desde el punto de vista del material (hardware) ya que se obtienen en la forma de circuitos integrados. Debido a la importancia de las compuertas NAND y NOR en el diseño de circuitos combinacionales, es importante poder reconocer la relación que existe entre

los circuitos construidos con compuertas AND-OR y sus diagramas NAND o NOR equivalentes.

La ejecución de los diagramas lógicos de dos niveles NAND y NOR fue presentada en la Sección 3-6. Aquí se considera el caso más general de los circuitos de multinivel. El procedimiento para obtener circuitos NAND se presenta en esta sección y para los circuitos NOR en la siguiente sección.

Compuerta universal

La compuerta NAND se conoce como la compuerta universal ya que cualquier sistema digital se puede configurar con ella. Los circuitos combinacionales y secuenciales pueden construirse también con esta compuerta ya que el circuito flip-flop (el elemento de memoria usado más frecuentemente en los circuitos secuenciales) puede construirse a partir de dos compuertas NAND conectadas especialmente como se muestra en la Sección 6-2.

Para demostrar que cualquier función de Boole puede configurarse con compuertas NAND, se necesita no solamente mostrar que las operaciones lógicas AND, OR y NOT pueden ser configuradas con compuertas NAND. La configuración de las operaciones AND, OR y NOT con compuertas NAND se muestra en la Figura 4-10. La operación NOT se obtiene de una compuerta NAND de una sola entrada, lo cual constituye otro símbolo para el inversor. La operación AND requiere dos compuertas NAND. La primera produce la AND invertida y la segunda actúa como un inversor para producir la salida normal. La operación OR se logra mediante una compuerta NAND con inversores adicionales en cada entrada.

Una manera conveniente de configurar un circuito combinacional con compuertas NAND es obtener las funciones de Boole simplificadas en términos de AND, OR y NOT y convertir las funciones a lógica NAND. La con-

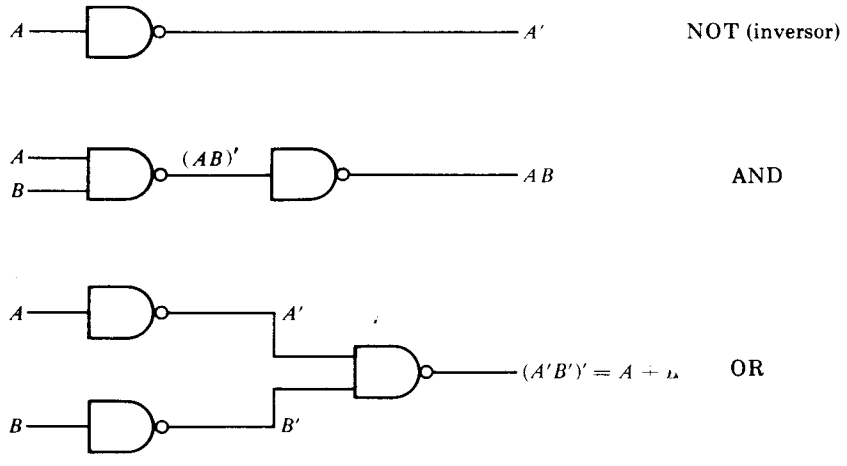


Figura 4-10 Configuración del NOT, AND y OR por medio de compuertas NAND

versión de expresiones algebraicas de operaciones AND, OR, NOT a operaciones NAND son comúnmente muy complicadas ya que envuelve un gran número de aplicaciones del teorema de De Morgan. La dificultad se elude mediante el uso de manipulaciones de circuitos y reglas sencillas las cuales se esbozan a continuación:

Configuración de las funciones de Boole — Método del diagrama de bloque

La configuración de funciones de Boole con compuertas NAND pueden obtenerse por medio de una técnica de manipulación del diagrama de bloque. Este método requiere que se dibujen otros dos diagramas lógicos antes de obtener el diagrama lógico NAND. Sin embargo el procedimiento es muy simple y directo:

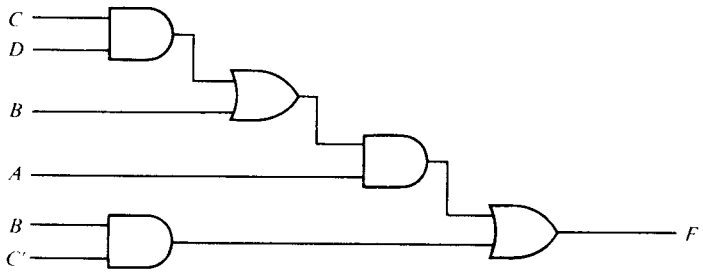
1. A partir de una expresión algebraica, dibújese el diagrama lógico con compuertas AND, OR y NOT. Asúmase que se tienen disponibles las entradas normales y sus compuertas.
2. Dibújese un segundo diagrama lógico con la lógica NAND equivalente, como se da en la Figura 4-10 y sustitúyase para cada compuerta AND, OR y NOT.
3. Quítese cualquier par de inversores en cascada del diagrama ya que la doble inversión no produce una función lógica. Quítese los inversores conectados a entradas externas simples y complemén-tese la variable de entrada correspondiente. El nuevo diagrama lógico obtenido es la configuración con compuertas NAND requerido.

Este procedimiento se ilustra en la Figura 4-11 para la función:

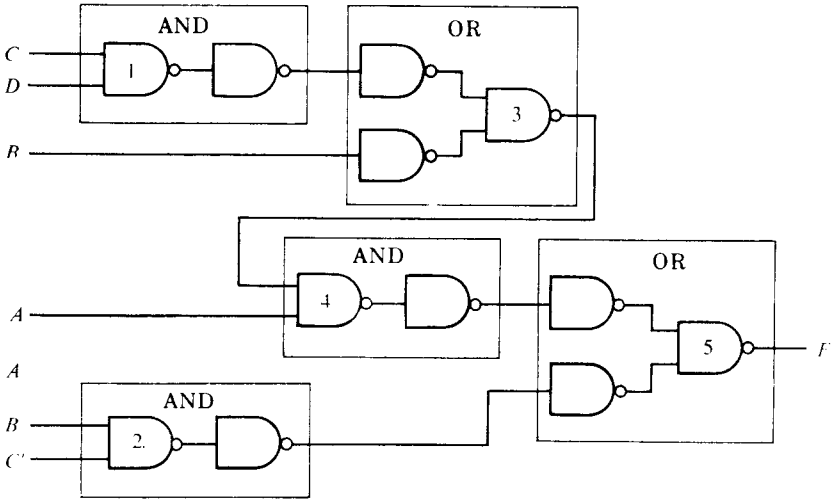
$$F = A(B + CD) + BC'$$

La ejecución AND-OR de esta función se muestra en el diagrama lógico de la Figura 4-11(a). Para cada compuerta AND, se sustituye una compuerta NAND seguida de un inversor; para cada compuerta OR se sustituyen inversores de salida seguidos de una compuerta NAND. Esta sustitución se desprende directamente de las equivalencias lógicas de la Figura 4-10 y se muestra en el diagrama de la Figura 4-11(b). Este diagrama tiene siete inversores y cinco compuertas NAND de dos entradas con sus respectivos números dentro del símbolo de la compuerta. El par de inversores conectados en cascada (de cada recuadro AND a cada recuadro OR) se eliminan ya que forman doble inversión. El inversor conectado a la entrada B se quita y se asigna la variable de entrada como B'. El resultado es el diagrama lógico NAND mostrado en la Figura 4-11(c), con el número dentro de cada símbolo identificando la compuerta de la Figura 4-11(b).

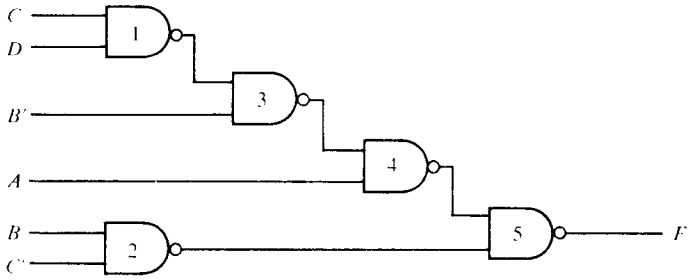
Este ejemplo demuestra que el número de compuertas NAND necesarias para ejecutar la función de Boole es igual al número de compuertas AND-OR si se cuenta con las entradas normales y su complemento. Si se



(a) Configuración AND-OR

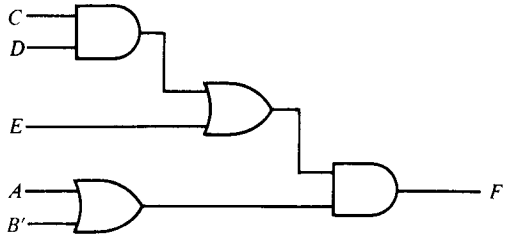


(b) Sustituyendo funciones NAND equivalentes de la Figura 5-8

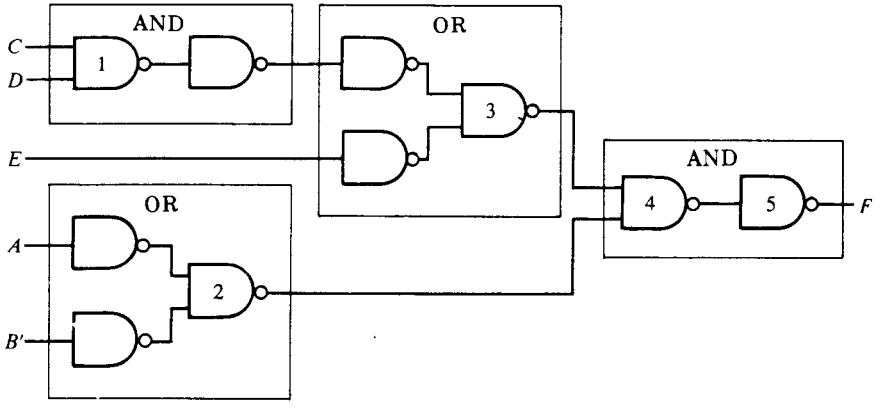


(c) Configuración con NAND

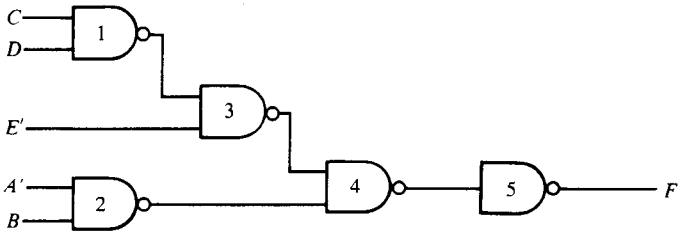
Figura 4-11 Configuración de $F = A(B + (CD)' + BC'$ con compuertas NAND



(a) Configuración AND-OR



(b) Sustituyendo funciones NAND equivalentes



(c) Configuración NAND

Figura 4-12 Configuración de $(A + B')(CD + E)$ con compuertas NAND

cuenta solamente con las entradas normales, se deben usar inversores para generar las entradas complementadas necesarias.

Un segundo ejemplo de configuración con NAND se muestra en la Figura 4-12. La función de Boole que se va a ejecutar es:

$$F = (A + B')(CD + E)$$

La configuración AND-OR se muestra en la Figura 4-12(a), y su sustitución con lógica NAND, en la Figura 4-12(b). Se pueden quitar un par de inver-

sores en cascada. Las tres entradas externas E, A y B' que van directamente a los inversores se complementan y se quitan los correspondientes inversores. La configuración final con compuertas NAND está en la Figura 4-12(c).

El número de compuertas NAND del segundo ejemplo es igual al número de compuertas AND-OR más un inversor adicional en la salida (compuerta NAND 5). En general, el número de compuertas NAND necesarias para configurar una función es igual al número de compuertas AND-OR, excepto por algún inversor ocasional. Esto es verdad si se cuenta con las entradas normales y su complemento ya que la conversión hace que se complementen ciertas variables de entrada.

El método del diagrama de bloque es algo aburrido de usar ya que requiere el dibujo de dos diagramas lógicos para obtener la respuesta en el tercero. Con alguna experiencia es posible reducir la cantidad de trabajo anticipándose a los pares de inversores en cascada y a los inversores en las entradas. Comenzando con el procedimiento esbozado, no es muy difícil derivar las reglas generales para la ejecución de funciones de Boole con compuertas NAND directamente de una expresión algebraica.

Procedimiento de análisis

El procedimiento anterior considera el problema de derivar un diagrama lógico NAND de una función de Boole dada. El proceso inverso es el análisis del problema que comienza con un diagrama lógico NAND dado y que culmina con una expresión de Boole o una tabla de verdad. El análisis de los diagramas lógicos NAND sigue el mismo procedimiento presentado en la Sección 4-6 para el análisis de los circuitos combinacionales. La única diferencia es que la lógica NAND requiere una aplicación repetida del teorema de De Morgan. Se demostrará la deducción de la función de Boole a partir de un diagrama lógico. Luego se demostrará la deducción de la tabla de verdad directamente del diagrama lógico NAND. Finalmente, se presentará un método para convertir un diagrama lógico NAND a un diagrama lógico AND-OR por medio de la manipulación de un diagrama de bloque.

Deducción de la función de Boole a partir de la manipulación algebraica

El procedimiento para deducir la función de Boole a partir de un diagrama lógico se esboza en la Sección 4-6. Este procedimiento se demuestra para el diagrama lógico NAND mostrado en la Figura 4-13, el cual es el mismo que aquel de la Figura 4-11(c). Primero, todas las salidas de las compuertas se marcan con símbolos aritméticos. Segundo se derivan de las funciones de Boole para las salidas de las compuertas que reciben solamente entradas externas:

$$T_1 = (CD)' = C' + D'$$

$$T_2 = (BC')' = B' + C$$

La segunda forma se desprende directamente del teorema de De Morgan y puede a veces ser más conveniente de usar. Tercero, las funciones de

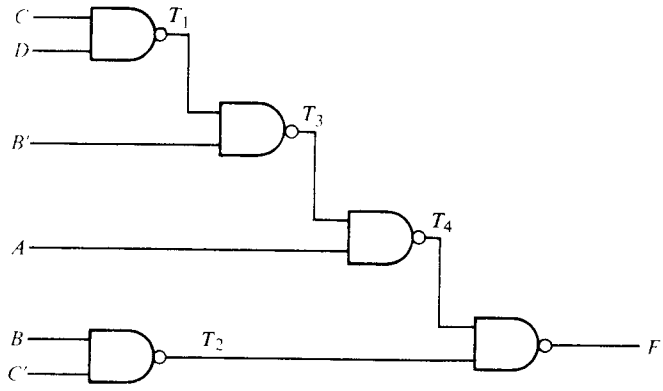


Figura 4-13 Ejemplo de análisis

Boole de compuertas que tienen entradas de funciones anteriormente derivadas se determinan en orden consecutivo hasta que la salida se exprese en términos de variables de entradas:

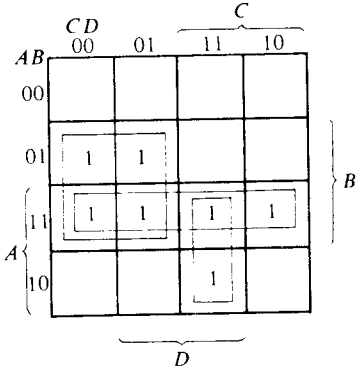
$$\begin{aligned} T_3 &= (B'T_1)' = (B'C' + B'D')' \\ &= (B + C)(B + D) = B + CD \\ T_4 &= (AT_3)' = [A(B + CD)]' \\ F &= (T_2T_4)' = \{(BC')'[A(B + CD)]'\}' \\ &= BC' + A(B + CD) \end{aligned}$$

Deducción de la tabla de verdad

El procedimiento para obtener la tabla de verdad directamente de un diagrama lógico se esboza en la Sección 4-6. Este procedimiento se demuestra por el diagrama lógico NAND de la Figura 4-13. Primero se listan las cuatro variables de entrada conjuntamente con las 16 combinaciones de unos y ceros como se muestra en la Tabla 4-3. Segundo se marcan las salidas de todas las compuertas con símbolos aritméticos como en la Figura 4-13. Tercero se obtienen las tablas de verdad para las salidas de aquellas compuertas que son función de las variables de entrada solamente. Estas son T_1 y T_2. $T_1 = (CD)'$, entonces se marcan ceros en aquellas filas donde ambas C y D sean iguales a 1 y se llena el resto de las filas de T_1 con unos. También $T_2 = (BC)'$ de tal manera que se marcan ceros en aquellas columnas donde $B = 1$ y $C = 0$ y se llena el resto de las filas de T_2 con unos. Seguidamente se procede a obtener la tabla de verdad para las salidas de aquellas compuertas que son función de las salidas definidas previamente hasta que se determine la columna para la salida F. Es posible, ahora, obtener una expresión algebraica a partir de la tabla de verdad derivada. El mapa mostrado en la Figura 4-14 se obtiene directamente de la Tabla 4-3 y tiene unos en los cuadrados de aquellos términos mínimos para los

Tabla 4-3 Tabla de verdad para el circuito de la Figura 4-13

A	B	C	D	T_1	T_2	T_3	T_4	F
0	0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	1	0	1



$F = AB + BC' + ACD$

Figura 4-14 Deducción de F a partir de la Tabla 4-3

cuales F es igual a 1. La expresión simplificada que se obtiene del mapa será:

$$F = AB + ACD + BC' = A(B + CD) + BC'$$

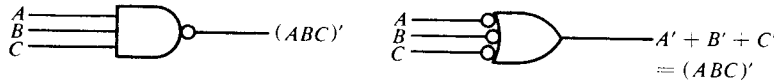
Esta es la misma expresión de la Figura 4-11, verificando así la respuesta correcta.

Trasformación del diagrama de bloque

Es conveniente algunas veces convertir un diagrama lógico NAND a su equivalente diagrama lógico AND-OR para facilitar el procedimiento de

análisis. Al hacer esto, la función de Boole puede derivarse muy fácilmente mediante el uso del teorema de De Morgan. La conversión de diagramas lógicos se logra a través del proceso inverso al usado para la ejecución de los mismos. En la Sección 3-6 se mostraron dos símbolos gráficos alternos para la compuerta NAND. Estos símbolos se repitieron en la Figura 4-15 por conveniencia. Por medio de un conciente uso de ambos términos, es posible convertir un diagrama NAND a una forma equivalente AND-OR.

La conversión de un diagrama lógico NAND a un diagrama AND-OR se logra a través de un cambio de símbolos de un AND invertido a OR invertido en niveles de compuertas *alternas*. El primer nivel que debe cambiarse a un símbolo OR invertido debe ser el último nivel. Estos cambios producen pares de círculos en la misma línea, los cuales pueden eliminarse ya que representan doble complementación. Una compuerta AND u OR de una sola entrada puede también quitarse ya que no hace ninguna función lógica. Una AND u OR de una sola entrada con un círculo en la entrada o la salida se cambia a un circuito inversor.



(a) AND invertido (b) OR invertido

Figura 4-15 Dos símbolos para una compuerta NAND

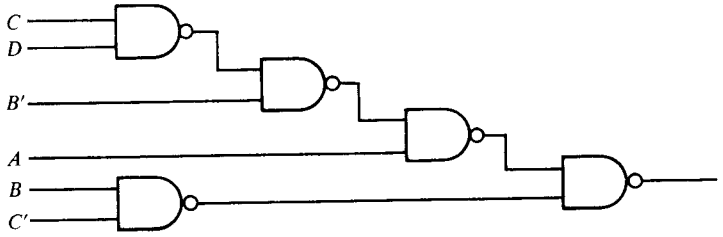
Este procedimiento se demuestra en la Figura 4-16. El diagrama lógico NAND de la Figura 4-16(a) se convierte a un diagrama AND-OR. El símbolo de la compuerta en el último nivel se cambia a un OR invertido. Observando los diferentes niveles, se encuentra otra compuerta que requiere un cambio de símbolo como se muestra en la Figura 4-16(b). Cualquier par de círculos en la misma línea se eliminan. Círculos que van a entradas externas se eliminan siempre y cuando la variable de entrada correspondiente esté complementada. El diagrama lógico AND-OR requerido se dibuja en la Figura 4-16(c).

4-8 CIRCUITOS NOR DE MULTINIVEL

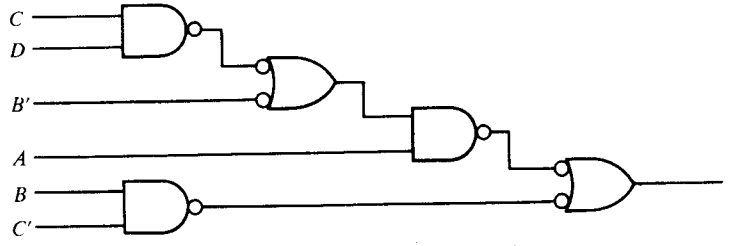
La función NOR es el dual de la función NAND. Por esta razón todos los procedimientos y reglas para la lógica NOR forman el dual de los correspondientes procedimientos y reglas desarrolladas para la lógica NAND. Esta sección enumera varios métodos para la configuración con lógica NOR y el análisis mediante el seguimiento de una lista de tópicos usados para la lógica NAND. Sin embargo no se incluye una explicación más detallada para prevenir repetición de lo expuesto en la Sección 4-7.

Compuerta universal

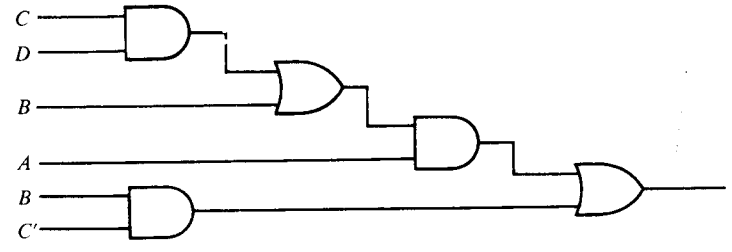
La compuerta NOR es universal ya que se puede ejecutar cualquier función de Boole con ella incluyendo el circuito flip-flop mostrado en la Sección 6-2. La conversión de AND, OR y NOT a NOR se muestra en la Figura 4-17.



(a) Diagrama lógico NAND



(b) Sustitución de símbolos OR invertido en niveles alternos



(c) Diagrama lógico AND-OR

Figura 4-16 Conversión de un diagrama lógico NAND a AND-OR

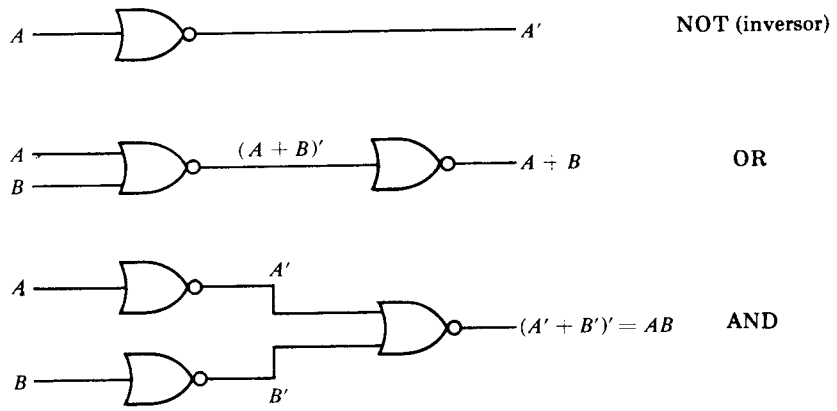


Figura 4-17 Configuración de NOT, OR y AND por medio de compuertas NOR

La operación NOT se obtiene de una compuerta NOR de una sola entrada lo que constituye otro símbolo para el inversor. La operación OR requiere dos compuertas NOR. La primera produce la OR invertida y la segunda actúa como un inversor para obtener la salida normal. La operación AND se logra por medio de la compuerta NOR con inversores adicionales en cada entrada.

Configuración de las funciones de Boole— Método del diagrama de bloque

El procedimiento del diagrama de bloque para configurar funciones de Boole con compuertas NOR es similar al procedimiento esbozado en la sección previa para las compuertas NAND.

1. Dibújese el diagrama lógico AND-OR a partir de una expresión algebraica. Asúmase que se cuenta con las entradas normales y su complemento.
2. Dibújese un segundo diagrama lógico con lógica NOR equivalente, de la manera usada en la Figura 4-17, sustituyendo cada compuerta AND, OR y NOT.
3. Elimínese los pares de inversores en cascada del diagrama. Quítese los inversores conectados a entradas externas sencillas y complemántese la variable de entrada correspondiente.

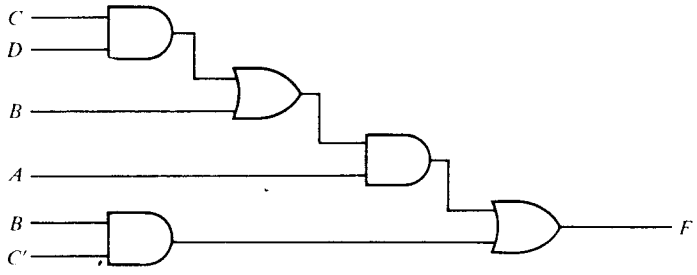
El procedimiento se ilustra en la Figura 4-18 para la función:

$$F = A(B + CD) + BC'$$

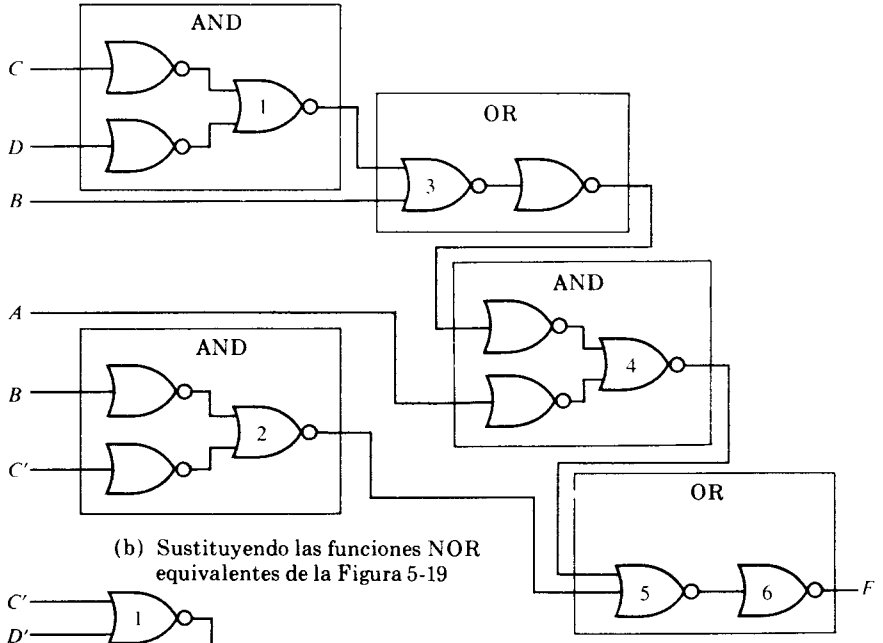
La ejecución AND-OR de la función se muestra en el diagrama lógico de la Figura 4-18(a). Por cada compuerta OR se sustituye una compuerta NOR seguida de un inversor. Por cada compuerta AND se sustituyen inversores en las entradas de una compuerta NOR. El par de inversores en cascada de la OR enmarcada y la AND enmarcada se elimina. Los cuatro inversores conectados a las entradas externas se remueven y se complementan las variables de entrada. El resultado es el diagrama lógico NOR mostrado en la Figura 4-18(c). El número de compuertas NOR en este ejemplo es igual al número de compuertas AND-OR más un inversor adicional a la salida (compuerta NOR 6). En general el número de compuertas NOR necesarias para la ejecución de funciones de Boole es igual al número de compuertas AND-OR excepto por un inversor ocasional. Lo anterior es válido siempre y cuando se cuente con las entradas normales y su complemento ya que la misma conversión induce a que se complementen ciertas variables.

Procedimiento de análisis

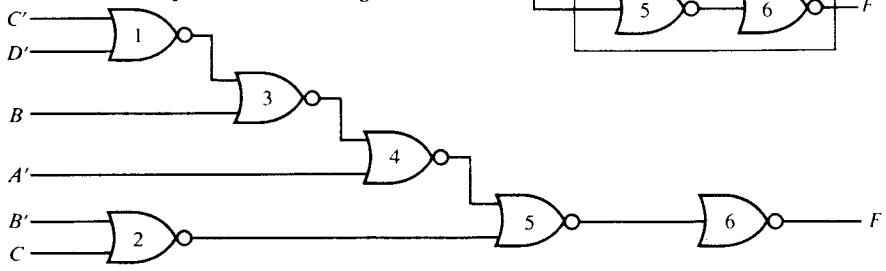
El análisis de los diagramas lógicos NOR sigue los mismos procedimientos presentados en la Sección 4-6 para el análisis de los circuitos combinacionales. Para deducir una función de Boole de un diagrama lógico se



(a) Configuración AND-OR



(b) Sustituyendo las funciones NOR equivalentes de la Figura 5-19



(c) Configuración NOR

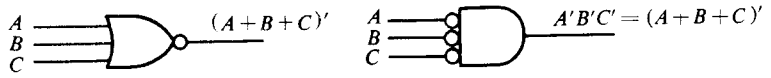
Figura 4-18 Configuración de $F = A(B + CD) + BC'$ con compuertas NOR

marcan las salidas de varias compuertas con símbolos arbitrarios. Mediante varias sustituciones se obtiene la variable de salida como función de las variables de entrada. Para obtener la tabla de verdad de un diagrama lógico sin primero deducir la función de Boole, se forma una tabla ha-

ciendo una lista de las n variables con 2^n filas de unos y ceros. La tabla de verdad de las salidas de las diferentes compuertas NOR se deducen en cadena hasta obtener la tabla de verdad de salida. La función de salida de una compuerta NOR típica es de la forma $T = (A + B' + C)'$, de tal manera que la tabla de verdad para T se marca con un 0 para aquellas combinaciones en que $A = 1$ ó $B = 0$ ó $C = 1$. El resto de las filas se llena con unos.

Trasformación del diagrama de bloque

Para convertir un diagrama lógico NOR a su equivalente diagrama lógico AND-OR, se usan los ímbolos para las compuertas NOR mostrados en la Figura 4-19. La OR invertida es el símbolo normal para una compuerta NOR y la AND invertida es una alternativa conveniente que utiliza el teorema de De Morgan y la convención de pequeños círculos en las entradas que denotan complementación.



(a) OR invertida (b) AND invertida

Figura 4-19 Dos símbolos para una compuerta NOR

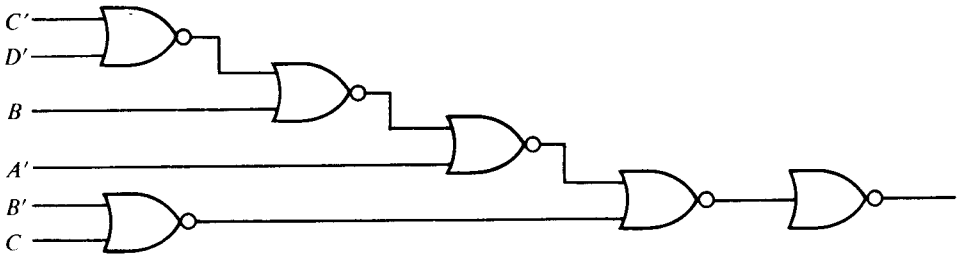
La conversión de un diagrama lógico NOR a un diagrama AND-OR se logra por medio de un cambio en los símbolos de OR invertida a AND invertida comenzando en el último nivel y en niveles alternos. Los pares de círculos pequeños en una misma línea se eliminan. Se quitan las compuertas AND u OR de una sola entrada a no ser que tengan un pequeño círculo a la salida o a la entrada, en cuyo caso se convierten en un inversor.

Este procedimiento se muestra en la Figura 4-20, donde el diagrama lógico NOR en (a) se convierte a un diagrama AND-OR. El símbolo para la compuerta en el último nivel (5) se cambia a un AND invertida. Al observar los diferentes niveles, se encuentra una compuerta en el nivel 3 y dos en el nivel 1. Estas compuertas sufren un cambio de símbolos como se muestra en (b). Cualquier par de círculos en una misma línea se remueven. Los círculos que van a entradas externas se quitan siempre y cuando se hayan complementado las variables de entrada correspondientes. La compuerta en el nivel 5 se convierte en una compuerta AND de una sola entrada y por tanto se elimina. El diagrama lógico AND-OR buscado, se muestra en la Figura 4-20(c).

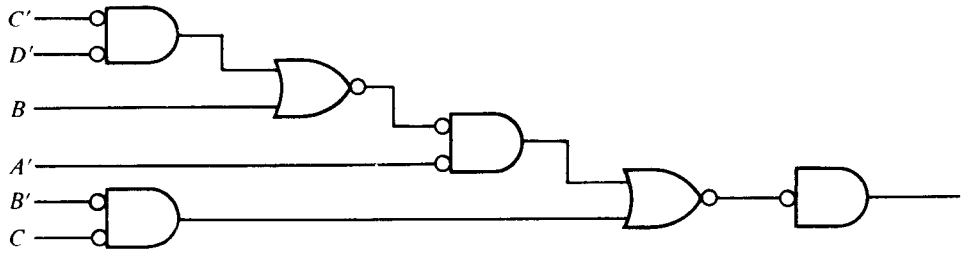
4-9 LAS FUNCIONES OR EXCLUSIVA Y DE EQUIVALENCIA

La OR-exclusiva y de equivalencia denotadas por $\oplus$ y $\odot$ respectivamente, son operaciones binarias que realizan las siguientes funciones de Boole:

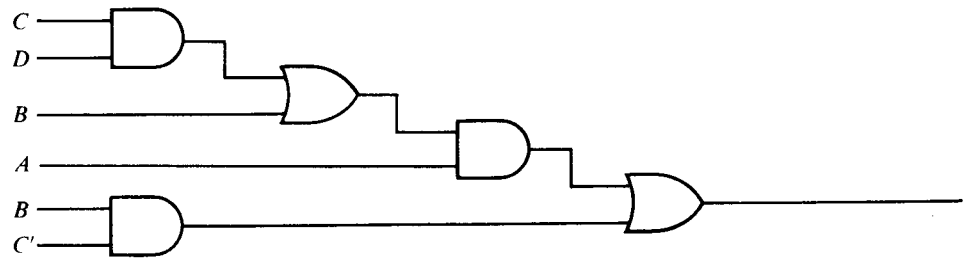
$$x \oplus y = xy' + x'y$$
$$x \odot y = xy + x'y'$$



(a) Diagrama lógico NOR



(b) Sustitución de símbolos AND invertida en niveles internos



(c) Diagrama lógico AND-OR

Figura 4-20 Conversión de un diagrama lógico NOR a AND-OR

Las dos operaciones son complementos entre sí. Cada una de ellas es asociativa y commutativa. Debido a las dos anteriores propiedades, una función de tres o más variables, puede expresarse sin paréntesis de la siguiente manera:

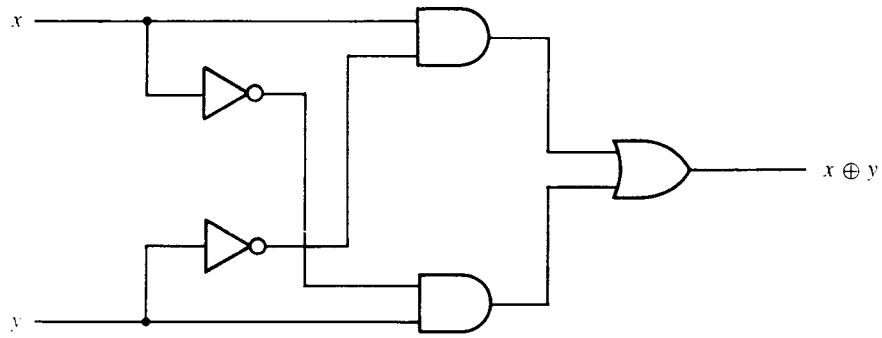
$$(A \oplus B) \oplus C = A \oplus (B \oplus C) = A \oplus B \oplus C$$

Esto implicaría la posibilidad de usar compuertas OR-exclusiva (o de equivalencia) con tres o más entradas. Sin embargo las compuertas OR-exclusiva de entrada múltiple son antieconómicas desde el punto de vista de los materiales. De hecho, aun la función de dos entradas se construye con otro tipo de compuertas. En la Figura 4-21(a), por ejemplo, se muestra la ejecución de la función OR-exclusiva de dos entradas con compuertas AND, OR y NOT. La Figura 4-21(b) la muestra con compuertas NAND.

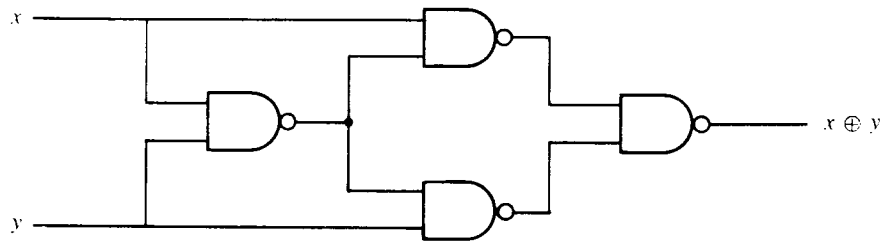
Solamente un número limitado de funciones de Boole se pueden expresar exclusivamente en términos de operaciones OR-exclusivas o de equivalencia. Empero, estas funciones resultan a menudo durante el diseño de sistemas digitales. Las dos funciones son particularmente útiles en operaciones aritméticas y en corrección de detección de errores.

Una expresión en OR-exclusiva de n variables es igual a una función de Boole con $2^n/2$ términos mínimos cuyos números binarios equivalentes tengan un número impar de unos. Esto se muestra en el mapa de la Figura 4-22(a) para el caso de cuatro variables. Hay 16 términos mínimos para cuatro variables. La mitad de los términos mínimos tienen un valor numérico con un número impar de unos; la otra mitad tiene un valor numérico con un número par de unos. El valor numérico de un término mínimo se determina a partir de las filas y columnas de los cuadrados que representan el término mínimo. El mapa de la Figura 4-22(a) tiene unos en los cuadrados cuyos términos mínimos tienen un número impar de unos. La función puede expresarse en términos de operación OR-exclusiva con las cuatro variables. Lo anterior se justifica por medio de la siguiente manipulación algebraica:

$$\begin{aligned} A \oplus B \oplus C \oplus D &= (AB' + A'B) \oplus (CD' + C'D) \\ &= (AB' + A'B)(CD + C'D') + (AB + A'B')(CD' + C'D) \\ &= \Sigma(1, 2, 4, 7, 8, 11, 13, 14) \end{aligned}$$



(a) con compuertas AND-OR-NOT



(b) con compuertas NAND

Figura 4-21 Configuraciones del OR-exclusivo

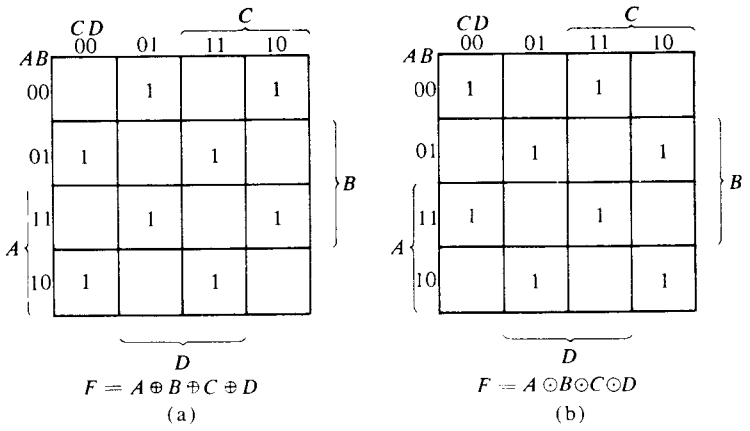


Figura 4-22 Mapa para cuatro variables (a) función OR-exclusiva y (b) función de equivalencia

Una expresión de equivalencia de n variables es igual a la función de Boole con $2^n/2$ términos mínimos cuyos números binarios equivalentes tienen un número par de ceros. Esto se demuestra en el mapa de la Figura 4-22(b) para el caso de cuatro variables. Los cuadrados, con unos representan los ocho términos mínimos con un número par de ceros y la función puede expresarse en términos de operaciones de equivalencia con las cuatro variables.

Cuando el número de variables en una función es impar, los términos mínimos con un número de par de ceros son los mismos que los términos con un número impar de unos. Esto se puede demostrar en el mapa de tres variables de la Figura 4-23(a). Por tanto, una expresión de OR-exclusiva es igual a una expresión de equivalencia cuando ambas tienen el mismo número impar de variables. Sin embargo, ellas forman los complementos entre sí cuando el número de variables es par de la manera como se muestra en los mapas de la Figura 4-22(a) y (b).

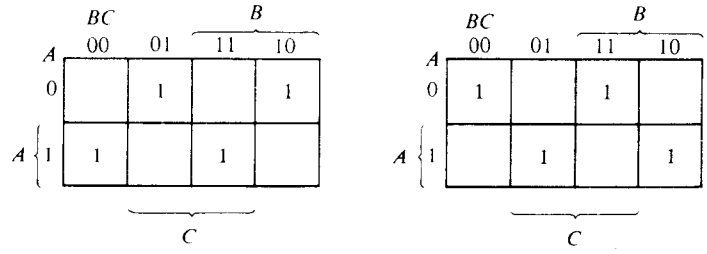
Cuando los términos mínimos de una función con un número impar de variables tiene un número par de unos (o por equivalencia a un número impar de ceros), la función puede expresarse como complemento de una expresión de OR-exclusiva o de equivalencia. Por ejemplo, la función de tres variables mostrada en el mapa de la Figura 4-23(b) puede expresarse de la siguiente manera:

$$(A \oplus B \oplus C)' = A \oplus B \odot C$$

o

$$(A \odot B \odot C)' = A \odot B \oplus C$$

La salida S de un sumador medio y la salida D de un sumador completo (Sección 4-3) puede configurarse con funciones OR-exclusivas ya que cada función consiste en cuatro términos mínimos con valores numéricos que tienen un número impar de unos. La función de OR-exclusiva se usa



(a) $F = A \oplus B \oplus C = A \odot B \odot C$ (b) $F = A \oplus B \odot C = A \odot B \oplus C$

Figura 4-23 Mapa para funciones de tres variables

bastante en la ejecución de operaciones aritméticas digitales debido a que estas últimas se ejecutan por medio de un proceso que requiere una operación de sumas o restas repetitivas

Las funciones de OR-exclusiva y de equivalencia son muy útiles en sistemas que requieren códigos de detección y corrección de errores. Como se trató en la Sección 1-6, un bit de paridad es una forma de detectar errores durante la trasmisión de información binaria. Un bit de paridad es un bit extra incluido con un mensaje binario para hacer el número de unos par o impar. El mensaje, incluyendo el bit de paridad, se trasmite y luego se comprueba en el extremo de recepción los errores. Un error se detecta si la paridad comprobada no corresponde a la trasmitida. El circuito que genera el bit de paridad en un trasmisor se llama *generador de paridad;* el circuito que comprueba la paridad en el receptor se llama *comprobador de paridad.*

Como ejemplo, considérese un mensaje de tres bits para trasmitirse con un bit de paridad impar. La Tabla 4-4 muestra la tabla de verdad para el generador de paridad. Los tres bits x, y y z constituyen el mensaje y son las entradas al circuito. El bit de paridad P es la salida. Para una paridad impar, el bit P se genera para hacer el número total de unos impar (P incluido). De la tabla de verdad, se ve que $P = 1$ cuando el número de unos en x, y y z es par. Esto corresponde al mapa de la Figura 4-23(b); así, la función P puede expresarse de la siguiente manera:

$$P = x \oplus y \odot z$$

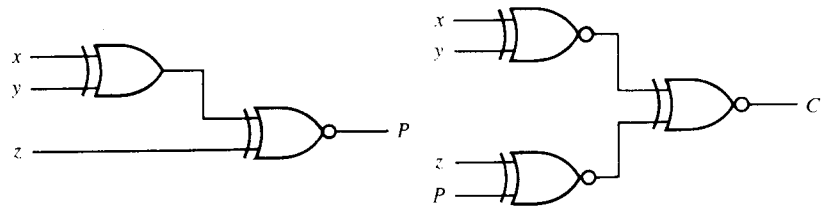
El diagrama lógico para el generador de paridad se muestra en la Figura 4-24(a). Este consiste en una compuerta OR-exclusiva de dos entradas y una compuerta de equivalencia de dos entradas. Las dos compuertas pueden ser intercambiadas y aun producir la misma función ya que P es igual a:

$$P = x \odot y \oplus z$$

El mensaje de tres bits y el bit de paridad se trasmiten a su destino donde se aplican a un circuito de observación de paridad. Durante la trasmisión ocurre un error si la paridad de los cuatro bits es impar, ya que la información binaria trasmitida fue originalmente impar. La salida C del comprobador de paridad debe ser un 1 cuando ocurre un error, es

Tabla 4-4 Generación de paridad impar.

Mensaje de tres bits			Bit de paridad generado
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



(a) Generador de paridad impar de tres bits

(b) Comprobador de paridad impar de cuatro bits

Figura 4-24 Diagramas lógicos para la generación y comprobación de la paridad

decir, cuando el número de unos en las cuatro entradas sea par. La Tabla 4-5 es la tabla de verdad de un circuito comprobador de paridad impar. De él se observa que la función de C consiste de ocho términos mínimos con valores numéricos que tienen un número par de ceros. Esto corresponde al mapa de la Figura 4-22(b): de tal manera que la expresión puede ser expresada con operadores de equivalencia de la siguiente manera:

$$C = x \odot y \odot z \odot P$$

El diagrama lógico de un comprobador de paridad se muestra en la Figura 4-24(b) y consiste en tres compuertas de equivalencia de dos entradas.

Vale la pena anotar que el generador de paridad puede ejecutarse con el circuito de la Figura 4-24(b) si la entrada P se mantiene permanentemente en lógica 0 y la salida se marca P, la ventaja estriba en el hecho de que ambos circuitos pueden ser usados para generación de paridad y comprobación.

Es obvio del presente ejemplo que los circuitos de generación y comprobación de paridad tengan una función de salida que incluye la mitad de los términos mínimos cuyos valores numéricos tengan un número par o impar de unos. En consecuencia estos se pueden ejecutar con compuertas de equivalencia y de OR-exclusiva.

Tabla 4-5 Comprobación de la paridad impar

Cuatro bits recibidos				Comprobación del error-paridad
x	y	z	P	C
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

REFERENCIAS

1. Rhyne, V. T., *Fundamentals of Digital Systems Design.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1973.
2. Peatman, J. P., *The Design of Digital Systems.* Nueva York: McGraw-Hill Book Co., 1972.
3. Nagle, H. T. Jr., B. D. Carrol, y J. D. Irwin, *An Introduction to Computer Logic.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1975.
4. Hill, F. J., y G. R. Peterson, *Introduction to Switching Theory and Logical Design,* 2a. ed. Nueva York: John Wiley & Sons, Inc., 1974.
5. Maley, G. A., y J. Earle, *The Logic Design of Transistor Digital Computers.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1963.
6. Friedman, A. D., y P. R. Menon, *Theory and Design of Switching Circuits.* Woodland Hills, Calif.: Computer Science Press, Inc., 1975.

PROBLEMAS

4-1. Un circuito combinacional tiene cuatro entradas y una salida. La salida es igual a 1 cuando (1) todas las entradas sean iguales a 1 o (2) ninguna de las entradas sea igual a 1 o (3) un número impar de entradas sea igual a 1.

(a) Obtenga la tabla de verdad.
(b) Encuentre la función de salida simplificada en suma de productos.
(c) Encuentre la función de salida simplificada en producto de sumas.
(d) Dibuje los dos diagramas lógicos.

4-2. Diseñe un circuito combinacional que acepte un número de tres bits y genere un número binario de salida igual al cuadrado del número de entrada.

4-3. Es necesario multiplicar dos números binarios, cada uno de dos bits para formar su producto en binarios. Asuma los dos números representados por a_1, a_0 y b_1, b_0 donde el suscrito 0 denote el bit menos significativo.
(a) Determine el número de líneas de salida necesarias.
(b) Encuentre las expresiones de Boole simplificadas para cada salida.

4-4. Repita el Problema 4-3 para formar la suma (en vez del producto) de los dos números binarios.

4-5. Diseñe un circuito combinacional con cuatro líneas de entrada que representen un dígito decimal en BDC y cuatro líneas de salida que generan el complemento de 9 del dígito de entrada.

4-6. Diseñe un circuito combinacional cuya entrada es un número de cuatro bits y cuya salida es el complemento de 2 del número de entrada.

4-7. Diseñe un circuito combinacional que multiplique por 5 una entrada en dígito decimal representada en BDC. La salida debe ser también en BDC. Demuestre que las salidas pueden obtenerse de las líneas de entrada sin usar ninguna compuerta lógica.

4-8. Diseñe un circuito combinacional que detecte un error en la representación de un dígito decimal en BDC. En otras palabras obtenga un diagrama lógico cuya salida sea lógica 1 cuando las entradas tengan una combinación poco usual en el código.

4-9. Configure un sustractor completo con dos sustractores medios y una compuerta OR.

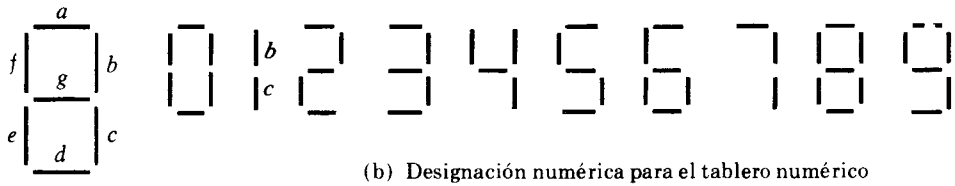
4-10. Demuestre cómo un sumador completo puede ser convertido a un sustractor completo con la adición de un circuito inversor.

4-11. Diseñe un circuito combinacional que convierta un dígito decimal del código 8,4, – 2, – 1 a BDC.

4-12. Diseñe un circuito combinacional que convierta un dígito decimal del código 2,4,2,1 al código 8,4, – 2, – 1.

4-13. Obtenga el diagrama lógico que convierte un número binario de cuatro dígitos a un número decimal en BDC. Nótese que se necesitan dos dígitos decimales ya que los números binarios van de 0 a 15.

4-14. Un decodificador BDC a siete segmentos es un circuito combinacional que acepta un número decimal en BDC y genera las salidas apropiadas para la selección de segmentos en un indicador usado para mostrar el dígito decimal. Las siete salidas del decodificador (*a, b, c, d, e, f, g*), seleccionan los segmentos correspondientes en el indicador como se muestra en la Figura P4-14 (a). La designación numérica escogida para representar el número decimal se muestra en la Figura P4-14(b). Diseñe el circuito decodificador de BDC a siete segmentos.



(a) Designación de segmentos

(b) Designación numérica para el tablero numérico

Figura P4-14

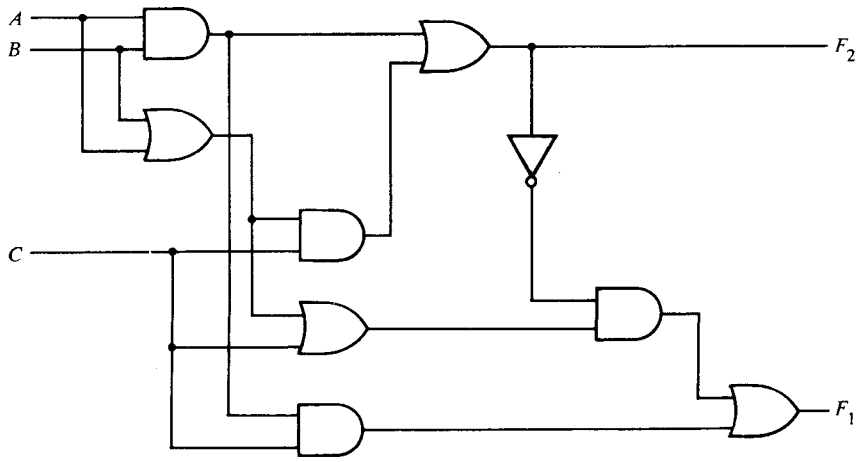


Figura P4-15

4-15. Analice los dos circuitos combinacionales mostrados en la Figura P4-15. Obtenga las funciones de Boole para las dos salidas y explique la operación del circuito.

4-16. Deduzca la tabla de verdad del circuito mostrado en la Figura P4-15.

4-17. Mediante el uso del diagrama de bloque, convierta el diagrama lógico de la Figura 4-8 a una configuración con NAND.

4-18. Repita el Problema 4-17 para una configuración con NOR.

4-19. Obtenga el diagrama lógico NAND de un sumador completo de las funciones de Boole.

$$C = xy + xz + yz$$

$$S = C'(x + y + z) + xyz$$

4-20. Determine la función de Boole para la salida F del circuito de la Figura P4-20. Obtenga un circuito equivalente con menos compuertas NOR.

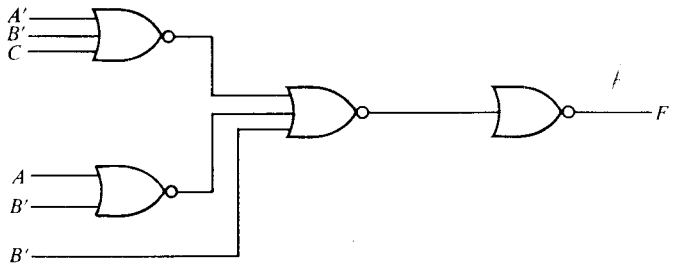
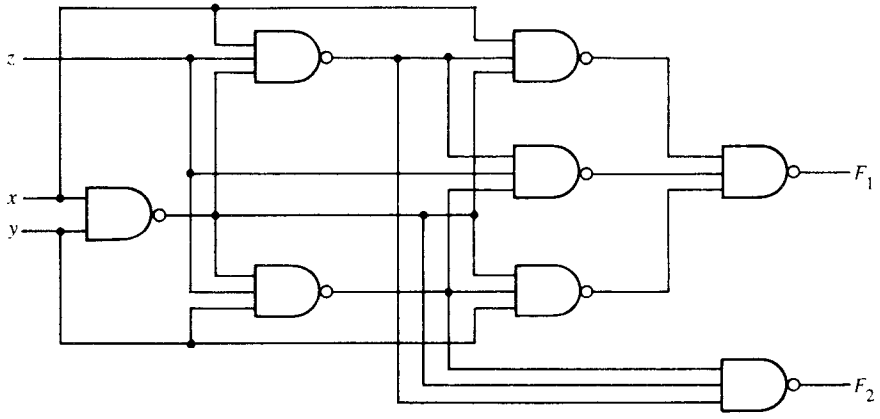


Figura P4-20

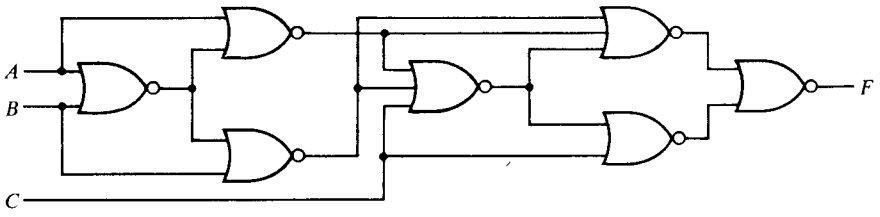
4-21. Determine las funciones de Boole de salida de los circuitos en la Figura P4-21.

4-22. Obtenga la tabla de verdad para los circuitos en la Figura P4-21.

4-23. Obtenga el diagrama lógico equivalente AND-OR de la Figura P4-21(a).



(a)



(b)

Figura P4-21

4-24. Obtenga el diagrama lógico equivalente AND-OR de la Figura P4-21(b).

4-25. Obtenga el diagrama lógico de una función de equivalencia de dos entradas usando (a) compuertas AND, OR y NOT: (b) compuertas NOR y (c) compuertas NAND.

4-26. Demuestre que el circuito en la Figura 4-21(b) es una OR-exclusiva.

4-27. Demuestre que $A \odot B \odot C \odot D = \Sigma(0, 3, 5, 6, 9, 10, 12, 15)$.

4-28. Diseñe un circuito combinacional que convierta un número de cuatro bits en código reflejado (Tabla 1-4) a un número binario de cuatro bits. Ejecute el circuito con compuertas OR-exclusiva.

4-29. Diseñe un circuito combinacional para comprobar la paridad par de cuatro bits. Se requiere una salida de lógica 1 cuando los cuatro bits no constituyen una paridad par.

4-30. Ejecute las cuatro funciones de Boole listadas usando los tres circuitos sumadores medios (Figura 4-2e).

$D = A \oplus B \oplus C$

$E = A'BC + AB'C$

$F = ABC' + (A' + B')C$

$G = ABC$

4-31. Ejecute la función de Boole:

$$F = AB'CD' + A'BCD' + AB'C'D + A'BC'D$$

con compuertas OR-exclusiva y AND.

Lógica combinacional con MSI y LSI

5

5-1 INTRODUCCION

El propósito de la simplificación de las funciones de Boole es obtener una expresión algebraica que cuando se configure resulte en un circuito de bajo costo. Sin embargo, el criterio que determina un circuito de bajo costo o sistema debe definirse si se va a evaluar el éxito de la simplificación lograda. El procedimiento de diseño para los circuitos combinacionales presentado en la Sección 4-2 minimiza el número de las compuertas necesarias para ejecutar una función dada. Este procedimiento clásico asume que, dados dos circuitos que realizan la misma función, aquel que utilice menos compuertas es preferible debido a que cuesta menos. Esto no es necesariamente cierto cuando se usan circuitos integrados.

Como se incluyen varias compuertas lógicas en una sola pastilla de CI se vuelve económico usar la mayoría de las compuertas de una pastilla utilizada aunque al hacerlo se aumente el total de compuertas. Más aún, algunas de las interconexiones entre las compuertas en muchos CI son internas a la pastilla y es más económico usar tantas interconexiones internas posibles para poder minimizar el número de conexiones entre pastillas externas. Con los circuitos integrados, no es la cantidad de compuertas lo que determinan el costo, sino el número y tipo de CI usado y el número de interconexiones externas necesarias para ejecutar una función dada.

Hay numerosas ocasiones cuando el método clásico de la Sección 4-2 no produce el mejor circuito combinacional para ejecutar una función dada. Además, la tabla de verdad y el procedimiento de simplificación en este método se vuelve muy complicado, si el número de variables de entrada es excesivamente grande. El circuito final obtenido dice si debe ser configurado con una conexión aleatoria de compuertas SSI, las cuales podrían utilizar un número relativamente grande de CI y cableado de interconexión. En la mayoría de los casos la aplicación de un procedimiento de diseño alterno puede producir un circuito combinacional para una función dada aun mejor que el obtenido al seguir el método de diseño clásico. La posibilidad de un procedimiento de diseño alterno depende de un problema particular y del ingenio del diseñador. El método clásico constituye un procedimiento general tal, que si se usa se garantiza que se producen resul-

tados. Sin embargo, cuando se amplía el método clásico es aconsejable investigar la posibilidad de un método alterno que sea más eficiente para el problema particular entre manos.

La primera pregunta que debe contestarse antes de pasar por un diseño detallado de un circuito combinacional, es si la función está disponible en una pastilla de CI. La mayoría de circuitos MSI se obtienen comercialmente. Estos circuitos realizan funciones digitales específicas comúnmente usadas en el diseño de sistemas de computadores digitales. Si no se encuentra un componente MSI que produzca exactamente la función necesaria, un diseñador recursivo debe poder formular un método para incorporar un MSI en un circuito. La selección de componentes MSI con preferencia sobre las compuertas SSI es extremadamente importante ya que invariablemente dará como resultado una reducción considerable de pastillas de CI y de cables de interconexión.

La primera mitad de este capítulo presenta ejemplos de circuitos combinacionales diseñados por métodos diferentes a los procedimientos clásicos. Todos los ejemplos demuestran la construcción interna de las funciones MSI existentes. Así se presentan nuevas herramientas de diseño y al mismo tiempo se familiariza el lector con las funciones MSI existentes. Es muy importante conocer las funciones MSI existentes no solamente en el diseño de circuitos combinacionales, sino también en el diseño de sistemas de computadores digitales más complicados.

Ocasionalmente se encuentran circuitos MSI y LSI que pueden aplicarse directamente al diseño y ejecución de cualquier circuito combinacional. Cuatro técnicas de diseño de lógica combinacional mediante MSI y LSI se introducen en la segunda mitad de este capítulo. Estas técnicas hacen uso de las propiedades generales de los decodificadores, multiplexores, memorias de programación (ROM) y arreglos lógicos programables (PLU). Estos cuatro componentes de CI tienen un gran número de aplicaciones. Su uso en la configuración de circuitos combinacionales descritos aquí es una de las muchas aplicaciones.

5-2 SUMADOR PARALELO BINARIO

El sumador completo introducido en la Sección 4-3 forma la suma de dos bits y un bit de arrastre previo. Dos números binarios de n bits pueden sumarse por medio de este circuito. Para demostrar con un ejemplo específico considérese dos números binarios, $A = 1011$ y $B = 0011$ cuya suma $S = 1110$. Cuando se agregan un par de bits de un sumador completo el circuito produce un bit de arranque que se usa con el par de bits de una posición más significativa. Esto se muestra en la siguiente tabla:

Suscrito i	4 3 2 1		*Sumador completo de la Figura 4-5*
Arrastre de entrada	0 1 1 0	C_i	z
Sumando	1 0 1 1	A_i	x
Sumando	0 0 1 1	B_i	y
Suma	1 1 1 0	S_i	S
Arrastre de salida	0 0 1 1	C_{i+1}	C

Los bits se suman con sumadores completos, comenzando con el bit menos significativo (suscrito) para formar el bit de suma y el bit de arrastre. Las entradas y las salidas del circuito sumador completo de la Figura 4-5 se indican a continuación. El arrastre de entrada C_1 en la posición menos significativa debe ser 0. El valor de C_{i+1} en una posición significativa dada es el arrastre de salida del sumador completo. Este valor se trasfiere al bit de arrastre de entrada del sumador completo que agrega los bits a una posición significativa de mayor posición a la izquierda. La suma de bits es generada así, comenzando desde la posición de la extrema derecha y es disponible tan pronto como se genere el bit de arrastre previo correspondiente.

La suma de dos números binarios de n bits, A y B pueden generarse de dos maneras: en serie o en paralelo. El método de la suma en serie usa solamente un circuito sumador completo y un elemento acumulador para conservar el arrastre de salida generado. El par de bits en A y B se trasfiere en serie, uno a la vez a través del solo sumador completo para producir una cadena de bits salida de la suma. El bit de arrastre de salida acumulado de un par de bits se usa como bit de arrastre de entrada para el siguiente par de bits. El método en paralelo usa n circuitos sumadores completos y todos los bits de A y B se aplican simultáneamente. El bit de arrastre de salida de un sumador completo se conecta al arrastre de entrada del sumador completo de la posición siguiente a la izquierda. Una vez se hayan generado los bits de arrastre, los bits de la suma correcta salen por las salidas de suma de los sumadores completos.

Un *sumador paralelo binario* es una función digital que produce una suma aritmética de dos números binarios en paralelo. Este consiste en sumadores completos conectados en cascada con la salida de arrastre de un sumador completo conectado al arrastre de entrada del siguiente sumador completo.

La Figura 5-1 muestra la interconexión de cuatro circuitos sumadores completos (FA) para dar un sumador paralelo binario de cuatro bits. Los sumadores de A y los bits sumadores de B se designan por medio de números suscritos de derecha a izquierda con el suscrito 1 denotando el bit de más bajo orden. Los arrastres se conectan en cadena a través de los sumadores completos. El arrastre de entrada del sumador es C_1 y la salida de arrastre es C_5. Las salidas S generan los bits de suma requeridos. Cuando el circuito sumador completo de cuatro bits se encapsula dentro de una pastilla CI tendrá cuatro terminales para un sumando, cuatro terminales para otro sumando, cuatro terminales para los bits de suma y dos terminales para los arrastres de entrada y salida.*

Un sumador completo de n bits requiere n sumadores completos. Puede construirse a partir de las CI sumadores completos de 4, 2 y 1 bit conectando en cascada varias pastillas. La salida de arrastre de una pastilla debe conectarse a la entrada de arrastre de aquella con los siguientes bits de mayor orden.

Los sumadores completos de 4 bits son un ejemplo típico de una función MSI. Pueden usarse en muchas aplicaciones que incluyen operaciones aritméticas. Obsérvese que el diseño de este circuito por medio del mé-

*Un ejemplo de un sumador completo de cuatro bits es el CI TTL tipo 74283.

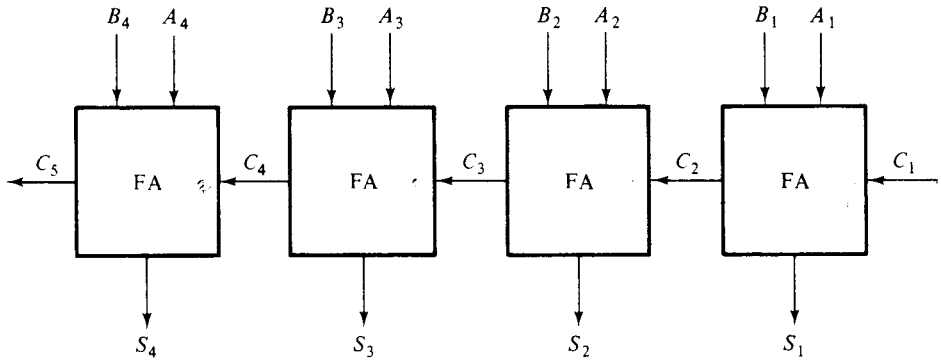


Figura 5-1 Sumadores completos de 4 bits

todo clásico necesitaría una tabla de verdad con $2^9 = 512$ entradas, ya que hay nueve entradas al circuito. Mediante el uso de un método iterativo de colocar en cascada una función ya conocida se puede obtener una configuración simple y bien organizada.

La aplicación de esta función MSI al diseño de un circuito combinacional se demuestra con el siguiente ejemplo:

> ***EJEMPLO 5-1:*** Diséñese un conversor de código BDC a exceso 3.
>
> Este circuito fue diseñado en la Sección 4-5 por medio del método clásico. El circuito obtenido de este diseño se muestra en la Figura 4-8 y requiere 11 compuertas. Cuando se ejecuta con compuertas SSI requiere 3 circuitos integrados y 14 conexiones internas (sin incluir las conexiones de entrada y de salida). La inspección de las tablas de verdad revela que el código equivalente de exceso 3 puede obtenerse del código BDC mediante la suma del binario 0011. Esta suma puede ejecutarse fácilmente mediante el circuito MSI de sumadores completos de 4 bits mostrado en la Figura 5-2. El dígito BDC se aplica a las entradas A, las entradas B se colocan a 0011 constante. Esto se logra aplicando lógica 1 a B_1 y B_2 y lógica 0 a B_3, B_4 y C_1. La lógica 1 y la lógica 0 son señales físicas cuyos valores dependen de la clase de familia de los CI usados. Para los circuitos TTL, lógica 1 equivale a 3,5 voltios y lógica 0 equivale a tierra. Las salidas S del circuito darán el código equivalente de exceso 3 del dígito de entrada en BDC. Esta configuración requiere un CI y 5 conexiones, sin incluir las conexiones de entrada y salida.

Propagación del arrastre

La suma de dos números binarios en paralelo implica que todos los bits de los sumandos están disponibles para el cálculo al mismo tiempo. Como en cualquier circuito combinacional, la señal debe propagarse por las com-

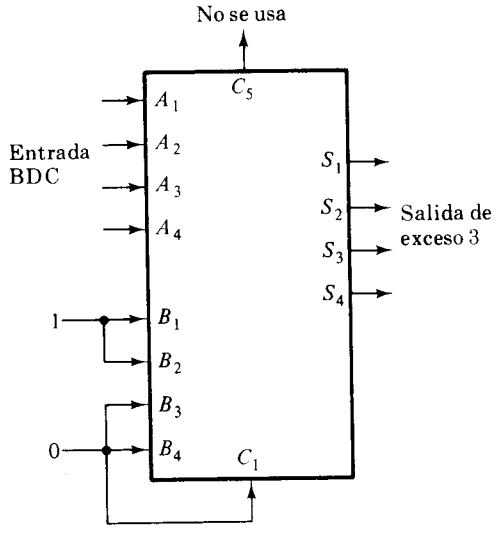


Figura 5-2 Convertidor de código de BDC a exceso 3

puertas antes que la suma de salida correcta esté disponible en los terminales de salida. El tiempo de propagación total es igual al retardo de propagación de una compuerta típica multiplicando por el número de niveles de compuertas en el circuito. El mayor tiempo de propagación en un sumador paralelo es el tiempo que se toma el bit de arrastre en propagarse por los sumadores completos. Como cada bit de la salida de suma depende del valor del arrastre de entrada, el valor de S_i en cualquier estado dado en el sumador, estará en su valor final estable solamente hasta que el bit de arrastre de entrada a este estado se haya propagado. Considérese la salida S_4 en la Figura 5-1. Las entradas A_4 y B_4 alcanzan un valor estable tan pronto como las señales de entrada se apliquen al sumador. Pero la entrada de arrastre C_4 no va a su estado estable final hasta que esté disponible C_3 en su valor de estado estable. De manera similar, C_3 tiene que esperar a C_2 y así sucesivamente hasta C_1. Así, irán la salida S_4 y el arrastre C_5 a un valor final de estado estable hasta que se propague el arrastre a través de todos los estados.

El número de niveles de compuertas para la propagación del arrastre se puede deducir del circuito del sumador completo. Este circuito es deducido en la Figura 4-5 y redibujado en la Figura 5-3 por conveniencia. Las variables de entrada y salida usan el suscrito i para denotar un estado típico de un sumador paralelo. Las señales en P_i y G_i llegan a su valor de estado estable después de la propagación por sus compuertas respectivas. Estas dos señales son comunes a todos los sumadores completos y dependen solamente de los bits de entrada de los sumandos. La señal del arrastre de entrada, C_i, se propaga al arrastre de salida, C_{i+1} a través de una compuerta AND y una compuerta OR, lo cual constituye dos niveles de compuertas. Si hay cuatro sumadores completos en el sumador paralelo, la salida de arrastre C_5 tendrá $2 \times 4 = 8$ niveles de compuertas desde C_1 hasta C_5. El tiempo de propagación total en el sumador será el tiempo de

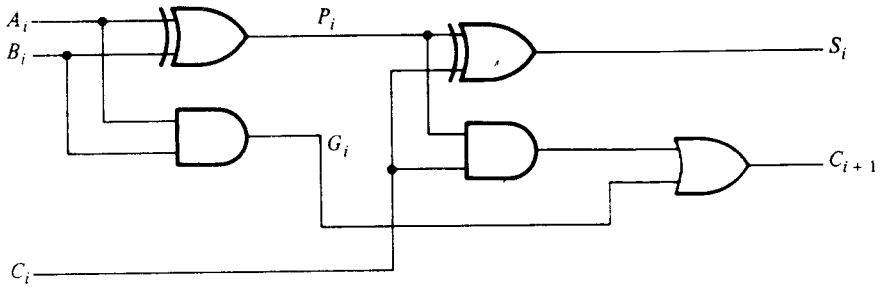


Figura 5-3 Circuito sumador completo

propagación en un sumador medio, más ocho niveles de compuertas. Para un sumador paralelo de n bits, hay 2 n niveles de compuertas para el bit de arrastre por los cuales se debe propagar.

El tiempo de propagación del arrastre es un factor limitante de la velocidad con la cual se suman dos números en paralelo. Aunque un sumador paralelo, o un circuito convencional, tengan siempre un valor en sus terminales de salida, las salidas no serán las correctas si no se les da a las señales el tiempo suficiente para propagarse a través de las compuertas conectadas desde las entradas hasta las salidas. Como todas las operaciones aritméticas se ejecutan con sumas sucesivas, el tiempo comprendido durante el proceso de suma es muy crítico. Una solución obvia para reducir el tiempo de demora de propagación del arrastre es la de usar compuertas más rápidas con demoras reducidas a pesar de que los circuitos físicos tengan un límite de su capacidad. Otra solucion es la de aumentar la complejidad del equipo de tal manera que se reduzca el tiempo de demora del arrastre. Hay otras técnicas para reducir el tiempo de propagación del arrastre en un sumador paralelo. La técnica usada más extensamente emplea el principio de *observación del arrastre posterior* y se describe a continuación.

Considérese el circuito del sumador completo mostrado en la Figura 5-3. Si se definen dos variables binarias nuevas:

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

la suma de salida y el arrastre puede expresarse como:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

C_i se llama el *arrastre generado*, y produce un arrastre de salida cuando A_i y B_i son 1 sin tener en cuenta el arrastre de entrada. P_i se llama el *arrastre propagado* ya que es el término asociado con la propagación de C_i hasta C_{i+1}.

Se escribe la función de Boole para la salida de arrastre de cada estado y se sustituye para cada C_i su valor a partir de las ecuaciones previas:

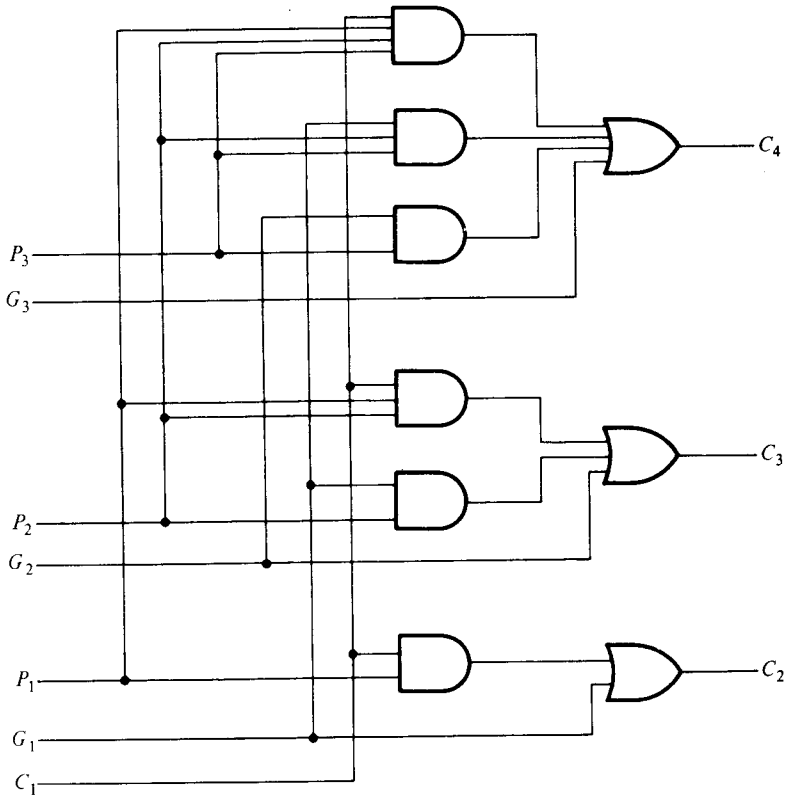


Figura 5-4 Diagrama lógico del generador del bit de arrastre posterior

$$C_2 = G_1 + P_1C_1$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2(G_1 + P_1C_1) = G_2 + P_2G_1 + P_2P_1C_1$$
$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_1$$

Como las funciones de Boole para cada arrastre de salida se expresan en suma de productos, cada función debe ser configurada con un nivel de compuertas AND seguidas de una compuerta OR (o mediante dos niveles de NAND). Las tres funciones de Boole para C_2, C_3 y C_4 se configuran con el generador del arrastre primario mostrado en la Figura 5-4. Nótese que C_4 no tiene que esperar a C_3 y C_2 para propagarse; de hecho C_4 se propaga al mismo tiempo que C_2 y C_3.*

La construcción de un sumador en paralelo de 4 bits con un arrastre posterior se muestra en la Figura 5-5. Cada salida de suma requiere dos compuertas OR-exclusivas. La salida de la primera OR-exclusiva genera la variable P_i y la compuerta AND genera la variable G_i. Todas las P y G se generan en dos niveles de compuertas. Los arrastres se propagan a través

*Un generador de arrastre posterior es el CI tipo 74182. Se compone de compuertas AND-OR invertida. Tiene también dos salidas para generar $C_5 = G + PC_1$.

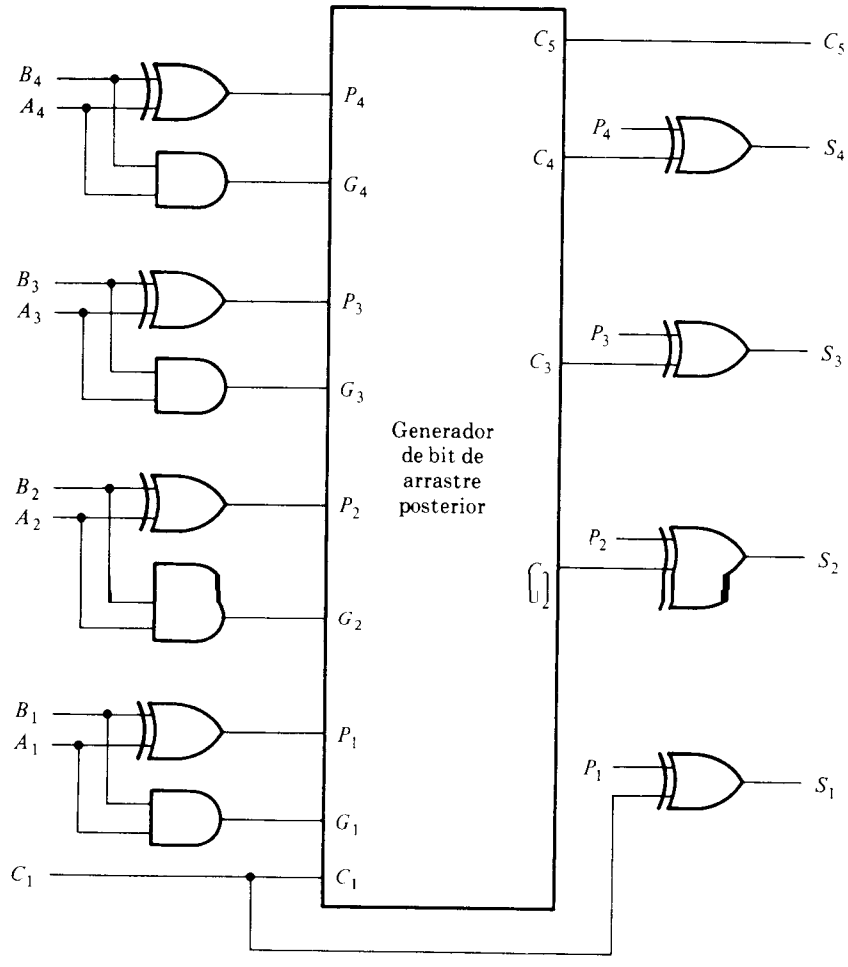


Figura 5-5 Sumadores completos de 4 bits con bit de arrastre posterior

del generador de arrastre posterior (similar al de la Figura 5-4) y se aplican como entradas a una segunda compuerta OR-exclusiva. Después que las señales P y G se establezcan a sus valores de estado estable, todos los arrastres de salida se generarán después de una demora de dos niveles de compuertas. Así, las salidas S_2 hasta S_4 tienen iguales tiempos de demora de propagación. El circuito de dos niveles para el arrastre de salida C_5 no se demuestra en la Figura 5-4. Este circuito puede derivarse fácilmente por el método de ecuación sustitución como se hizo anteriormente (ver Problema 5-4).

5-3 SUMADOR DECIMAL

Los computadores o calculadoras que realizan operaciones aritméticas directamente en el sistema de números decimales representan números decimales en la forma de binarios codificados. Un sumador para tal com-

putador debe usar circuitos aritméticos que aceptan números decimales codificados y presentan resultados en el código aceptado. Para suma binaria, fue suficiente considerar un par de bits significativos al tiempo, conjuntamente con el arrastre anterior. Un sumador decimal requiere un mínimo de nueve entradas y cinco salidas, ya que se requieren cuatro bits para codificar cada dígito decimal y el circuito debe tener un arrastre de entrada y uno de salida. Por supuesto, hay una gran variedad de circuitos de suma decimal que dependen del código usado para representar los dígitos decimales.

El diseño de un circuito combinacional de nueve entradas y cinco salidas por el método clásico requiere una tabla de verdad con $2^9 = 512$ entradas. La mayoría de las combinaciones de entrada son condiciones de no importa, ya que cada entrada de código binario tiene seis combinaciones que son válidas. Las funciones de Boole simplificadas por el circuito pueden obtenerse por un método de tabulado generado por un computador y el resultado podría ser probablemente una conexión de compuertas formando un patrón irregular. Un procedimiento alterno, es sumar los números con circuitos sumadores completos, teniendo en cuenta el hecho de que no se usan seis combinaciones en cada entrada de 4 bits. La salida debe ser modificada de tal manera que solamente aquellas combinaciones binarias, válidas del código decimal, se generen.

Sumador BDC

Considérese la suma aritmética de dos dígitos decimales en BDC, con un arrastre posible de un estado anterior. Como cada dígito de entrada no excede a la suma de salida no puede ser mayor que $9 + 9 + 1 = 19$, siendo el 1 en la suma, el arrastre de salida. Al suponer que se aplican dos dígitos BDC a un sumador binario de 4 bits, el sumador formará la suma en *binario* y producirá un resultado que puede variar entre 0 y 19. Estos números decimales se listan en la Tabla 5-1 y se marcan con símbolos K, Z_8, Z_4, Z_2 y Z_1. K es el arrastre y los suscritos bajo la letra Z representan los pesos 8, 4, 2 y 1 que deben ser asignados a los cuatro bits en el código BDC. La primera columna en la tabla lista las sumas binarias a medida que aparecen en las salidas de un sumador *binario* de 4 bits. La suma de salida de dos *dígitos decimales* debe representarse en BDC y debe aparecer en la forma listada en la segunda columna de la tabla. El problema es encontrar una regla simple por medio de la cual el número binario en la primera columna puede convertirse a la correcta representación de dígitos BDC del número en la segunda columna.

Al examinar el contenido de la tabla, es aparente que cuando la suma binaria sea igual o menor que 1001, el correspondiente número BDC es idéntico y por tanto no se necesita conversión. Cuando el número binario es mayor que 1001 se obtiene una representación BDC no válida. La suma del binario 6 (0110) a la suma binaria lo convierte a la representación BDC correcta y también produce el arrastre de salida requerido.

El circuito lógico que detecta la corrección necesaria puede derivarse de las entradas de la tabla. Es obvio que se necesita una corrección cuando

la suma binaria tiene un arrastre de salida $K = 1$. Las otras seis combinaciones desde 1010 hasta 1111 que necesitan una corrección tienen un 1 en la posición Z_8. Para distinguirlos del número binario 1000 y 1001 que también tienen un 1 en la posición Z_8, se especificará más adelante que Z_4 ó Z_2 deben tener un 1. La condición para que una corrección y un arrastre de salida pueda ser expresada por medio de una función de Boole:

$$C = K + Z_8Z_4 + Z_8Z_2$$

cuando $C = 1$, es necesario agregar 0110 a la suma binaria y suministrar un arrastre de salida a la siguiente etapa.

Tabla 5-1 Deducción de un sumador BDC

Suma binaria					Suma BDC					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

Un *sumador BDC* es un circuito que agrega dos dígitos BDC en paralelo y produce un dígito suma en BDC. Un sumador BDC debe incluir la corrección lógica en su construcción interna. Para agregar 0110 en la suma binaria, se usa un segundo sumador binario de 4 bits como se muestra en la Figura 5-6. Los dos dígitos decimales, conjuntamente con un arrastre de entrada, se agregan primero en el sumador binario de 4 bits superior para producir la suma binaria. Cuando el arrastre de salida es igual a cero, no se agrega nada a la suma binaria. Cuando es igual a 1 se agrega el binario 0110 a la suma binaria por medio del sumador binario de 4 bits infe-

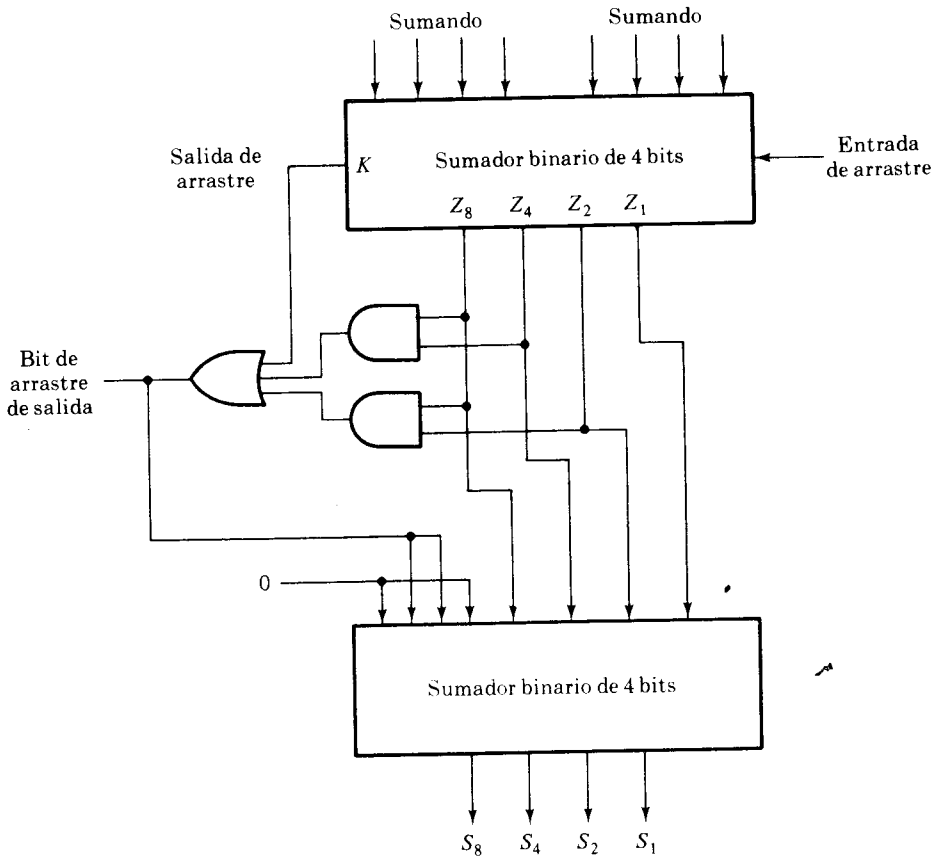


Figura 5-6 Diagrama de bloque de un sumador BDC

rior. El arrastre de salida generado a partir del sumador binario superior puede ignorarse porque da la información ya disponible en el terminal de arrastre de salida.

El sumador BDC puede construirse con tres CI. Cada uno de los sumadores de 4 bits es una función MSI y las tres compuertas para la lógica de corrección caben en una pastilla SSI. Sin embargo, el sumador BDC se obtiene en un circuito MSI.* Para alcanzar demoras de propagación más cortas, un sumador MSI BDC incluye los circuitos necesarios para los arrastres posteriores. El circuito sumador para la corrección no necesita todos los cuatro sumadores completos y este circuito puede optimizarse dentro de una pastilla de CI.

Un sumador paralelo decimal que suma n dígitos decimales necesita n etapas de sumadores BDC. El arrastre de salida de una etapa debe conectarse al arrastre de entrada de la siguiente etapa de mayor orden.

*El CI TTL tipo 82S83 es una sumador BDC.

5-4 COMPARADOR DE MAGNITUDES

La comparación de dos números es una operación que determina si un número es mayor que, menor que o igual a otro número. Un *comparador de magnitud* es un circuito combinacional que compara dos números, A y B y determina sus magnitudes relativas. El resultado de la comparación se especifica por medio de tres variables binarias que indican cuando $A > B$, $A = B$, ó $A < B$.

El circuito para comparar dos números de n bits tiene 2^{2n} entradas en la tabla de verdad y se vuelve muy complicado aun para $n = 3$. Por otra parte, como es de sospechar, un circuito comparador tiene cierta cantidad de regularidad. Las funciones digitales que poseen una regularidad inherente bien definida pueden diseñarse por medio de un procedimiento algorítmico si se encuentra su existencia. Un *algoritmo* es un procedimiento que especifica un conjunto finito de pasos, los cuales dan una solución al problema si se siguen. Se ilustrará este método deduciendo un algoritmo para el diseño de un comparador de magnitud de 4 bits.

El algoritmo es una aplicación directa de un procedimiento que usa una persona para comparar las magnitudes relativas de dos números. Considérese los números A y B cada uno con 4 dígitos y escríbase los coeficientes de los números en orden significativo descendente de la siguiente manera:

$$A = A_3A_2A_1A_0$$
$$B = B_3B_2B_1B_0$$

donde cada suscrito de letra representa uno de los dígitos del número. Los dos números son iguales si todos los pares de números significativos son iguales, es decir si $A_3 = B_3$ y $A_2 = B_2$ y $A_1 = B_1$ y $A_0 = B_0$. Cuando los números son binarios los dígitos son 1 ó 0 y la relación de igualdad para cada par de bits puede expresarse lógicamente con una función de equivalencia:

$$x_i = A_iB_i + A'_iB'_i \qquad i = 0, 1, 2, 3$$

donde $x_i = 1$ solamente si el par de bits en la posición i son iguales es decir si ambos son unos o ceros.

La igualdad de dos números A y B se indica en un circuito combinacional por medio de una variable binaria de salida que se designa con el símbolo $(A = B)$. Esta variable binaria es igual a 1 si los números de entrada A y B son iguales; de lo contrario será igual a 0. Para que exista esta condición de igualdad, todas las variables x_i deben ser iguales a 1. Esto produce una operación AND de todas las variables:

$$(A = B) = x_3x_2x_1x_0$$

la variable *binaria* $(A = B)$ es igual a 1 solamente si todos los pares de dígitos de los dos números son iguales.

Para determinar si A es mayor o menor que B se inspeccionan las magnitudes relativas de los pares de dígitos significativos comenzando desde la posición significativa más alta. Si los dos dígitos son iguales, se compara

el siguiente par de dígitos siguientes menos significativos. Esta comparación continúa hasta que se encuentre un par de dígitos desiguales. Si el correspondiente dígito de A es 1 y el de B es 0, se concluye que $A > B$. Si el correspondiente dígito de A es 0 y el de B es 1 se tiene que $A < B$. La comparación secuencial puede expresarse lógicamente por las dos siguientes funciones de Boole:

$$(A > B) = A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$$
$$(A < B) = A'_3B_3 + x_3A'_2B_2 + x_3x_2A'_1B_1 + x_3x_2x_1A'_0B_0$$

los símbolos $(A > B)$ y $(A < B)$ son variables de salida *binarias* que son iguales a 1 cuando $A > B$ ó $A < B$ respectivamente.

La ejecución con compuertas de las tres variables de salida derivadas es más simple de lo que parece ya que tiene cierta cantidad de repetición. Las salidas "desiguales" pueden usar las mismas compuertas que se necesitan para generar una salida "igual". El diagrama lógico del comparador de magnitud de 4 bits se muestra en la Figura 5-7.* Las cuatro x de salida se generan con circuitos de equivalencia (NOR-exclusiva) y se aplican a una compuerta AND para dar la variable binaria de salida $(A = B)$. Las otras dos salidas usan las variables x para generar las funciones de Boole listadas a continuación. Esta es una configuración de multinivel y como se puede ver tiene un patrón regular. El procedimiento para obtener circuitos comparadores de magnitud para números binarios de más de cuatro bits debe ser obvio para este ejemplo. El mismo circuito puede usarse para comparar las magnitudes relativas de dos dígitos BDC.

5-5 DECODIFICADORES

Cantidades discretas de información se presentan en sistemas digitales con códigos binarios. Un código binario de n bits es capaz de representar hasta 2^n elementos diferentes de información codificada. Un *decodificador* es un circuito combinacional que convierte la información binaria de n líneas de entrada a un máximo de 2^n líneas únicas de salida. Si la información decodificada de n bits tiene combinaciones no usadas o de no importa, la salida del decodificador tendrá menos de 2^n salidas.

Los decodificadores presentados aquí se llaman decodificadores en línea de n a m. En donde $m \leqq 2^n$. Su propósito es generar 2^n (o menos) términos mínimos de n variables de entrada. El nombre *decodificador* se usa conjuntamente con cierto tipo de convertidores de código tal como el decodificador BDC a siete segmentos (ver Problema 4-14).

Como ejemplo, considérese el circuito decodificador en línea de 3 a 8 de la Figura 5-8. Las tres entradas se decodifican en ocho salidas y cada salida representa uno de los términos mínimos de las variables de 3 entradas. Los tres inversores generan el complemento de las entradas y cada una de las ocho compuertas AND generan uno de los términos mínimos. Una

*El TTL tipo 7485 en un comparador de magnitud de 4 bits. Tiene tres entradas más para conectar los comparadores en cascada (ver Problema 5-14).

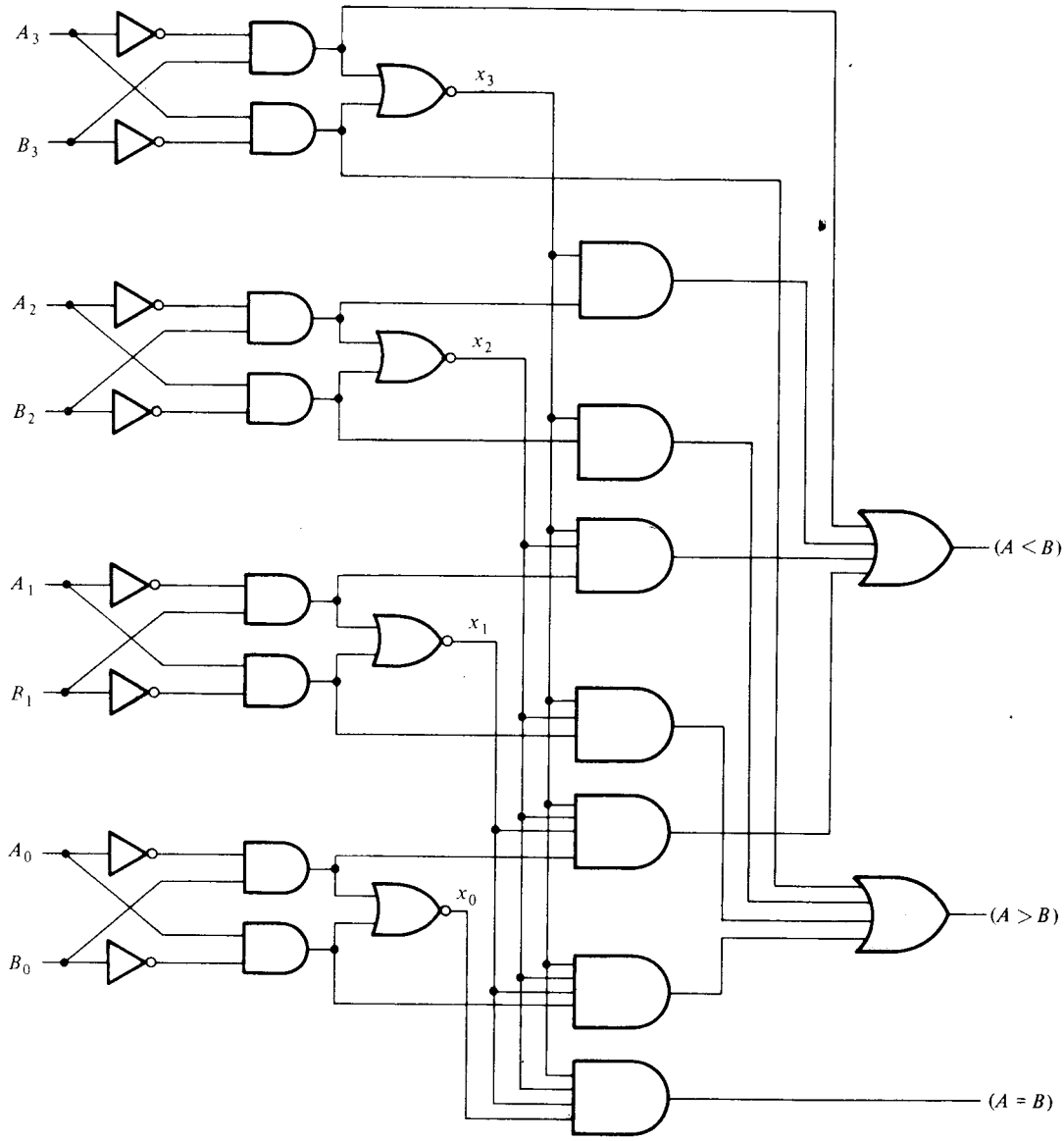


Figura 5-7 Comparador de magnitudes de 4 bits

aplicación particular de este decodificador sería una conversión binaria a octal. Las variables de entrada podrían representar un número binario y las salidas representarían los ocho dígitos en el sistema de numeración octal. Sin embargo un decodificador en línea de 3 a 8 puede ser usado para decodificar cualquier código de 3 bits para generar ocho salidas, una para cada elemento del código.

La operación del decodificador será clasificada más adelante a partir de las relaciones de entrada salida listadas en la Tabla 5-2. Obsérvese

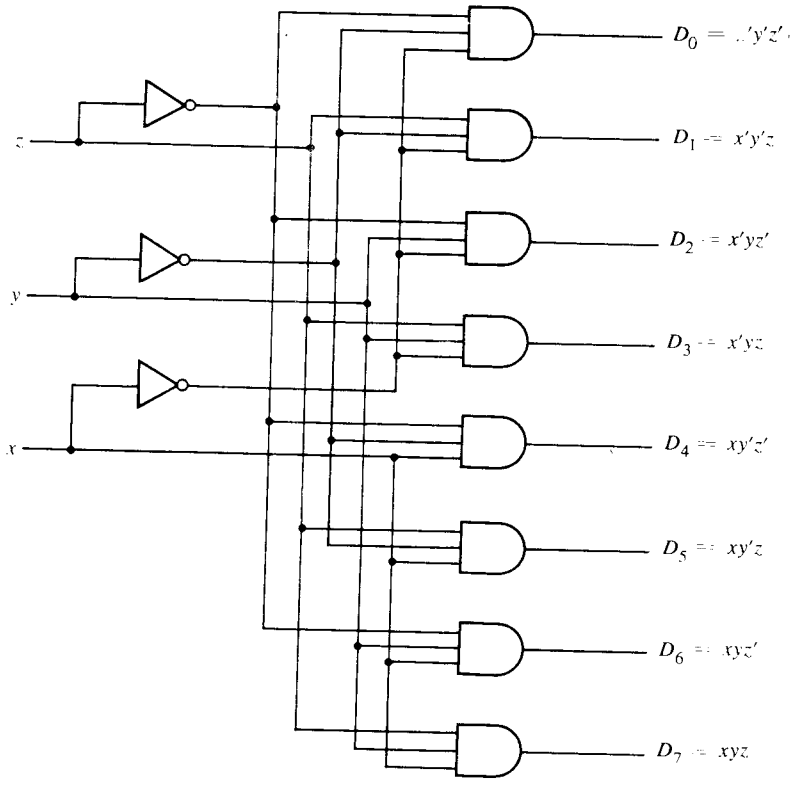


Figura 5-8 Decodificador en línea de 3 a 8

Tabla 5-2 Tabla de verdad del decodificador de línea de 3 a 8

Entradas			Salidas							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

que las variables de salida son mutuamente exclusivas ya que solamente una de las salidas es igual a 1 en cualquier momento. La línea de salida cuyo valor corresponde a 1 representa el término mínimo equivalente al número binario que se presenta en las líneas de entrada.*

EJEMPLO 5-2: Diseñar un decodificador BDC a decimal.

Los elementos de información en este caso son los diez dígitos decimales representados por el código BDC. El código en sí mismo tiene cuatro bits, por tanto, el decodificador debería tener cuatro entradas para aceptar el dígito codificado y las diez salidas para cada uno de los dígitos decimales. Esto dará un decodificador de 4 a 10 líneas de BDC a decimal.

No es necesario diseñar este decodificador ya que se puede encontrar en la forma de CI como una función MSI. De todas maneras se va a diseñar por dos razones: primero dará un conocimiento de lo que se debe esperar de tal función MSI; segundo, esto constituye un buen ejemplo para mostrar las consecuencias prácticas de las condiciones de no importa.

Como el circuito tiene diez salidas, sería necesario dibujar diez mapas para simplificar cada una de las funciones de salida. Hay seis funciones de no importa que deben considerarse para la simplificación de cada una de las funciones de salida. En vez de dibujar diez mapas, se dibujará solamente un mapa y se escribirán cada una de las variables de salida D_0 hasta D_9, dentro de su cuadrado de término mínimo, de la manera mostrada en la Figura 5-9. Hay seis combinaciones de entrada que nunca ocurren de tal manera que se marcan los cuadrados de los términos mínimos correspondientes con X.

Es responsabilidad del diseñador decidir cómo tratar las condiciones de no importa. Se asume que ha decidido usarlas de tal manera que se simplifican las funciones al número mínimo de

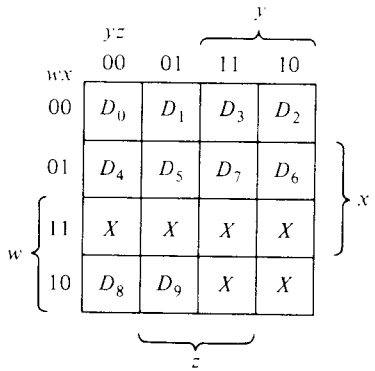


Figura 5-9 Mapa para simplificar un decodificador BDC a decimal

*El CI tipo 74138 es un decodificador en línea de 3 a 8. Se construye con compuertas NAND. Las salidas son los complementos de los valores mostrados en la Tabla 5-2.

literales. D_0 y D_1 no pueden combinarse con ningún término mínimo de no importa. D_2 puede combinarse con el término mínimo m_{10} de no importa para dar:

$$D_2 = x'yz'$$

El cuadrado con D_9 puede combinarse con otros tres cuadrados de no importa para dar:

$$D_9 = wz$$

Usando los términos de no importa para las otras salidas, se obtiene el circuito mostrado en la Figura 5-10. De esta manera los términos de no importa causan una reducción en el número de entradas en la mayoría de las compuertas AND.

Un diseñador cuidadoso debería investigar el efecto de la minimización anterior. A pesar de que bajo las condiciones de operación normal las seis combinaciones inválidas nunca ocurren. ¿Qué pasaría si hay una falla y ocurren? Un análisis del circuito de la Figura 5-10 muestra que las seis combinaciones no válidas de entrada producirán las salidas listadas en la Tabla 5-3. El lector puede mirar la tabla y decidir si el diseño es bueno o malo.

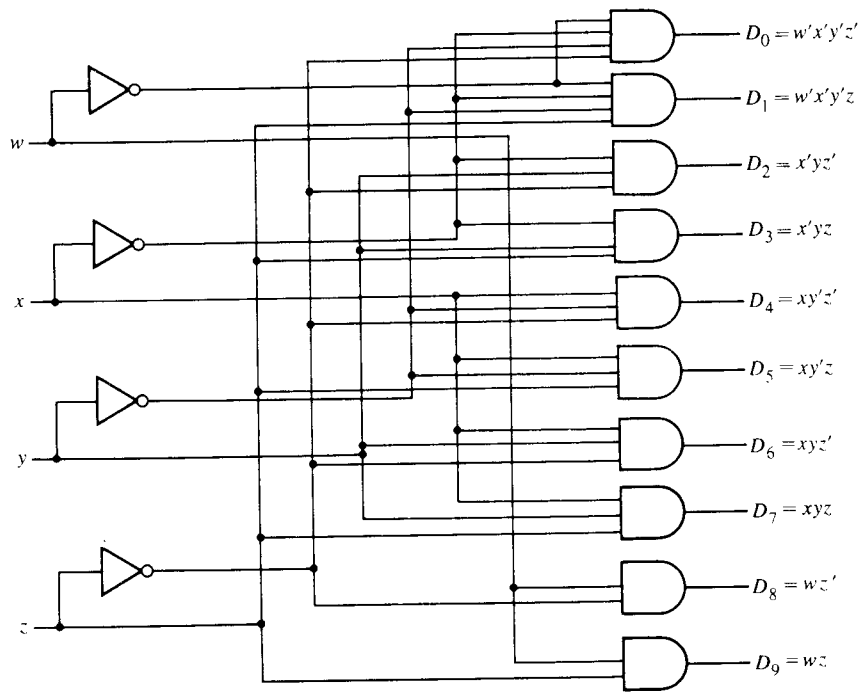


Figura 5-10 Decodificador BDC a decimal

Tabla 5-3 Tabla parcial de verdad para el circuito de la Figura 5-10

Entradas				Salidas									
w	x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
1	0	1	0	0	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	1	0	0	0	0	0	1
1	1	0	0	0	0	0	0	1	0	0	0	1	0
1	1	0	1	0	0	0	0	0	1	0	0	0	1
1	1	1	0	0	0	0	0	0	0	1	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1	0	1

Otra decisión de diseño razonable podría ser el hacer todas las salidas iguales a 0 cuando ocurre una combinación de entrada no válida.* Esto requeriría diez compuertas AND de cuatro entradas. Se deben considerar otras posibilidades pero de todas maneras no se deben tratar las condiciones de no importa indiscriminadamente, sino que se debe tratar de investigar su efecto una vez que el circuito esté en operación.

Configuración de circuitos con lógica combinacional

Un decodificador produce 2^n términos mínimos de n variables de entrada. Como cualquier función de Boole puede expresarse en suma de términos mínimos en la forma canónica, se puede usar un decodificador para generar los términos mínimos y una compuerta OR externa para formar la suma. De esta manera cualquier circuito combinacional con n entradas y m salidas puede configurarse con un decodificador en línea de n a 2^n y m compuertas OR.

El procedimiento para configurar un circuito combinacional por medio de un codificador y compuertas OR requiere que las funciones de Boole del circuito se expresen en suma de términos mínimos. Esta forma puede obtenerse fácilmente de la tabla de verdad o por expansión de las funciones a su suma de términos mínimos (ver Sección 2-5). Luego se escoge un decodificador que genere todos los términos mínimos de las n variables de entrada. Las entradas a cada compuerta OR se seleccionan de las salidas del decodificador de acuerdo a la lista de términos mínimos en cada función.

EJEMPLO 5-3: Construir un circuito sumador completo con un decodificador y dos compuertas OR.

De la tabla de verdad del sumador completo (Sección 4-3) se obtienen las funciones para este circuito combinacional en suma de términos mínimos:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z,) = \Sigma(3, 5, 6, 7)$$

*El CI tipo 7442 es un decodificador BDC a decimal. Las salidas seleccionadas están en el estado de 0 y todas las combinaciones inválidas darán una salida de solo unos.

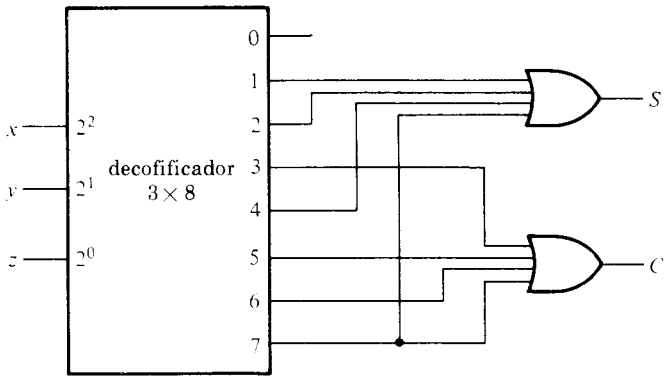


Figura 5-11 Configuración de un sumador completo a partir de un decodificador

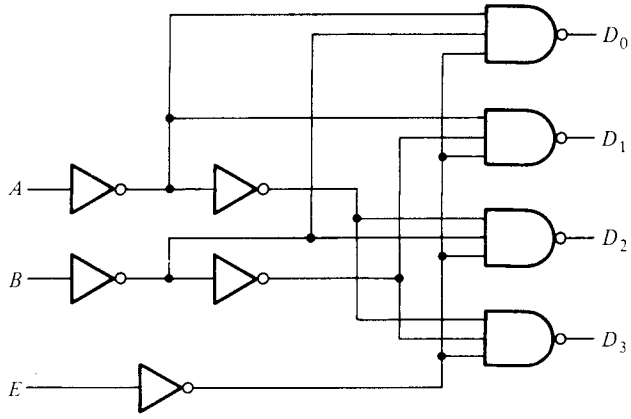
Como hay tres entradas y un total de ocho términos mínimos se necesita un decodificador en línea de 3 a 8. Su ejecución se muestra en la Figura 5-11. El decodificador genera los ocho términos mínimos de x, y, z. La compuerta OR para la salida S forma la suma de los términos mínimos 1, 2, 4 y 7. La compuerta OR para la salida C forma la suma de los términos mínimos 3, 5, 6 y 7.

Una función con una lista larga de términos mínimos requiere una compuerta OR con un gran número de entradas. Una función F que tiene una lista de k términos mínimos puede expresarse en forma de complemento F con $2^n - k$ términos mínimos. Si el número de términos mínimos de una función es mayor que $2^n/2$ entonces F' puede expresarse con menores términos mínimos que los que necesita F. En tal caso, es ventajoso usar una compuerta NOR para sumar los términos mínimos de F'. La salida de una compuerta NOR genera una salida normal F.

El método del decodificador se puede usar para ejecutar cualquier circuito combinacional. Sin embargo su realización se debe comparar con otras configuraciones posibles para determinar la mejor solución. En algunos casos este método podría dar la mejor combinación, especialmente si los circuitos combinacionales tienen muchas salidas y si cada función de salida (o su complemento) se expresa con una pequeña cantidad de términos mínimos.

Demultiplexores

Algunos CI se construyen con compuertas NAND. Como una compuerta NAND produce una operación AND con una salida invertida, es más económico generar los términos mínimos del decodificador en su forma complementada. La mayoría si no todos los CI decodificadores, incluyen una o más entradas de *activación* (enable), para controlar la operación del circuito. Un decodificador en línea de 2 a 4 con una entrada de activación y construido con compuertas NAND se muestra en la Figura 5-12. Todas las salidas son iguales a 1 si la entrada de activación E es 1, no importando los valores de las entradas A y B. Cuando la entrada de activación es igual a



(a) Diagrama lógico.

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

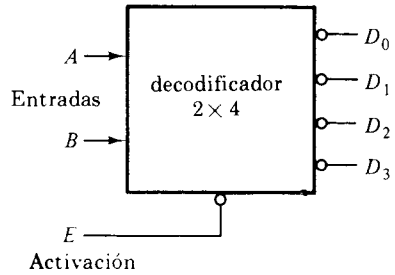
(b) Tabla de verdad

Figura 5-12 Un decodificador de línea 2 a 4 con entrada activadora (E)

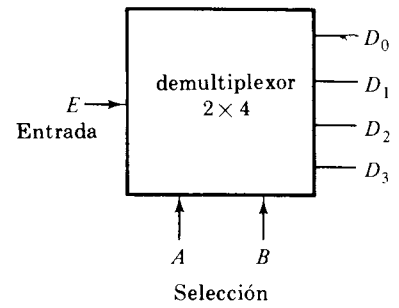
0, el circuito opera como decodificador con salidas complementadas. La tabla de verdad lista estas condiciones. Las X debajo de A y B son condiciones de no importa. La operación normal del decodificador ocurre solamente con $E = 0$ y las salidas se seleccionan cuando su estado es 0.

El diagrama de bloque del decodificador se muestra en la Figura 5-13(a). El circuito pequeño en la entrada E indica que el decodificador se activa cuando $E = 0$. El pequeño círculo a la salida indica que todas las salidas están complementadas.

Un decodificador con una entrada de habilitación puede funcionar como demultiplexor. Un *demultiplexor* es un circuito que recibe información por una sola línea y trasmite esta información en una de las 2^n líneas posibles de salida. La selección de una línea de salida específica se controla por los valores de los bits de n líneas de selección. El decodificador de la Figura 5-12 puede funcionar como demultiplexor si la línea E se toma como línea de entrada de datos y las líneas A y B como líneas de selección tal como se muestra en la Figura 5-13(b). La sola variable de entrada E



(a) Decodificador con activador



(b) Demultiplexor

Figura 5-13 Diagramas de bloque para el circuito de la Figura 5-12

tiene un camino a todas las salidas, pero la información de entrada se dirige solamente a una de las líneas de salida de acuerdo al valor binario de las dos líneas de selección A y B. Esto puede verificarse de la tabla de este circuito mostrada en la Figura 5-12(b). Por ejemplo si la selección de las líneas $AB = 10$ la salida D_2 tendrá el mismo valor que la entrada E, mientras que las otras salidas se mantienen en 1. Como las operaciones decodificador y demultiplexor se obtienen del mismo circuito, un decodificador con una entrada de activación se llama un *decodificador/demultiplexor*. Es la entrada de activación la que hace al circuito un demutiplexor; el decodificador de por sí puede usar compuertas AND, NAND y NOR.

Los circuitos decodificador/demultiplexor pueden conectarse conjuntamente para formar un circuito decodificador mayor. La Figura 5-14 muestra dos decodificadores de 3×8 con entradas activadoras conectadas para formar un decodificador de 4×16. Cuando $w = 0$, el decodificador superior se habilita y el otro se inhabilita. Las salidas del decodificador inferior son todas ceros y las ocho salidas superiores generan los términos mínimos 0000 a 0111. Cuando $w = 1$ se invierten las condiciones de habilitación; el decodificador inferior genera los términos mínimos 1000 a 1111, mientras que las salidas del decodificador superior son todas ceros. Este ejemplo demuestra la utilidad de las entradas activadoras de los CI. En general, las líneas activadoras son una característica conveniente para conectar dos o más CI con el propósito de expandir la función digital a una función similar con más entradas y salidas.

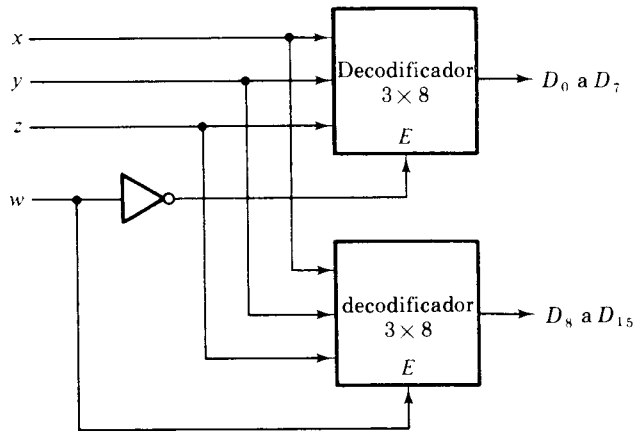


Figura 5-14 Un decodificador de 4×16 construido con dos decodificadores de 3×8

Codificadores

Un *codificador* es una función digital que produce una operación inversa a la del decodificador. Un codificador tiene 2^n (o menos) líneas de entrada y n líneas de salida. Las líneas de salida generan el código binario para las

2^n variables de entrada. Un ejemplo de un codificador se muestra en la Figura 5-15. El codificador octal a binario consiste en ocho entradas, una para cada uno de los ocho dígitos y tres salidas para generar el número binario correspondiente. Este se construye con compuertas OR, cuyas entradas se determinan a partir de la tabla de verdad dada en la Tabla 5-4. Los bits de salida de bajo orden z son 1 si los dígitos octales de entrada son impares. La salida y es 1 para los dígitos octales 2, 3, 6 ó 7. La salida x es 1 para los dígitos octales 4, 5, 6 ó 7. Nótese que D_0 no se conecta a ninguna compuerta OR; la salida binaria debe ser sólo ceros en este caso. Una salida de sólo ceros se obtiene también cuando todas las entradas sean cero. Esta discrepancia puede resolverse agregando una salida más para indicar el hecho de que todas las entradas no son ceros.

El codificador en la Figura 5-15 asume que solamente una línea de entrada puede ser igual a 1 en cualquier momento; de otra forma el circuito no tiene significado. Nótese que el circuito tiene ocho entradas y podría tener $2^8 = 256$ combinaciones de entrada posibles. Solamente ocho de estas combinaciones tienen significado. Las otras combinaciones son condiciones de no importa.

Los codificadores de este tipo (Figura 5-15) no se encuentran en CI ya que se pueden construir fácilmente con compuertas OR. El tipo de codificador que se encuentra en la forma de circuito integrado es el *codificador de prioridad.** Estos codificadores establecen una prioridad de entrada para asegurar que solamente la línea de entrada de la más alta prioridad se codifica. Así, en la Tabla 5-4, si la prioridad es dada a una entrada con un número suscrito mayor con respecto a un número suscrito menor, entonces si ambos D_2 y D_5 son lógica 1 simultáneamente, la salida será 101 porque D_5 tiene una mayor prioridad sobre D_2. Por supuesto, la tabla de verdad de un codificador de prioridad es diferente de la Tabla 5-4 (ver Problema 5-21).

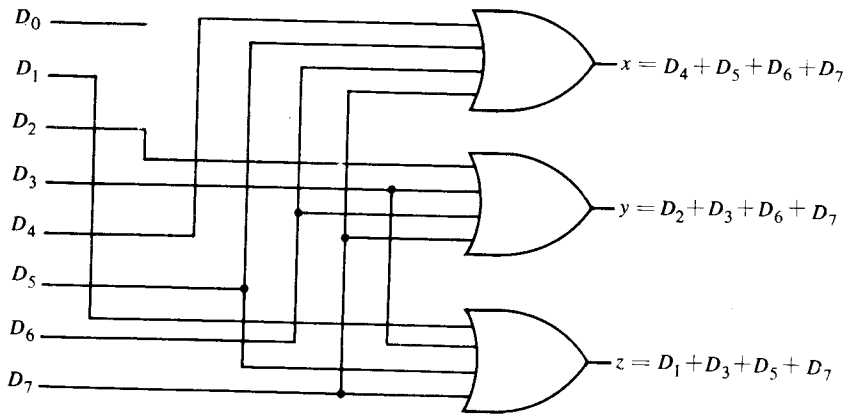


Figura 5-15 Codificador octal a binario

*Por ejemplo el CI tipo 74148.

Tabla 5-4 Tabla de verdad de codificador octal a binario

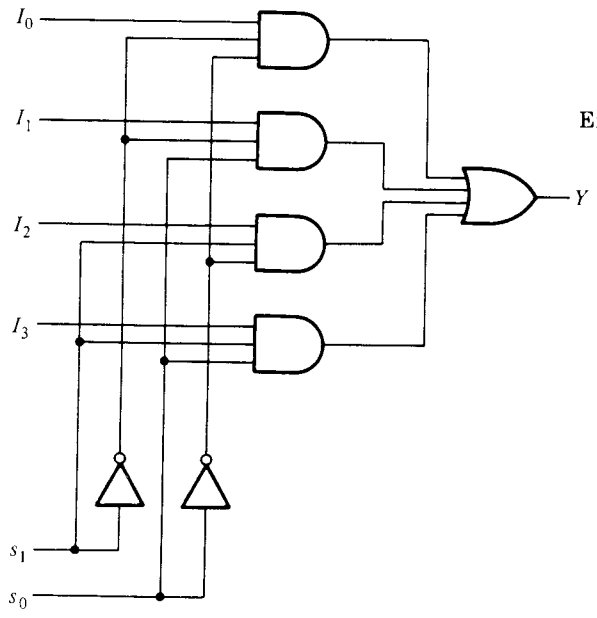
Entradas								Salidas		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

5-6 MULTIPLEXORES

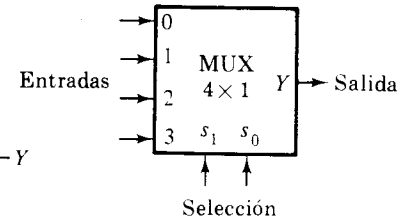
Multiplexar significa trasmitir una gran cantidad de unidades de información por un número pequeño de canales o líneas. Un *multiplexor digital* es un circuito combinacional que selecciona información binaria de una de muchas líneas de entrada para dirigirla a una sola línea de salida. La selección de una línea de entrada en particular es controlada por un conjunto de líneas de selección. Normalmente hay 2^n líneas de entrada y n líneas de selección cuyas combinaciones de bits determinan cuál entrada se selecciona.

Un multiplexor de 4 líneas a 1 línea se muestra en la Figura 5-16. Cada una de las cuatro líneas de entrada I_0 a I_3, se aplican a una entrada de una compuerta AND. Las líneas de selección s_1 y s_0 se decodifican para seleccionar una compuerta AND en particular. La tabla de función en la figura lista el camino de entrada a salida para cada combinación posible de bits de las líneas de selección. Cuando esta función MSI se usa en el diseño de un sistema digital ésta se representa en la forma de diagrama de bloque como se muestra en la Figura 5-16(c). Para demostrar la operación del circuito, considérese el caso cuando $s_1 s_0 = 10$. La compuerta AND asociada con la entrada I_2 tiene dos de sus entradas iguales a 1 y una tercera entrada conectada a I_2. Las otras tres compuertas AND tienen al menos una entrada igual a 0 lo cual hace su salida igual a 0. La salida de la compuerta OR es ahora igual al valor de I_2 generando así un camino de la entrada seleccionada a la salida. Un multiplexor se llama también un *selector de datos* ya que selecciona una de muchas entradas y guía la información binaria a la línea de salida.

Las compuertas AND y los inversores en un multiplexor se asemejan a un circuito decodificador y sin embargo ellos decodifican las líneas de selección de entrada. En general, un multiplexor de 2^n a 1 línea se construye con un decodificador de n a 2^n agregándole 2^n líneas de entrada, cada una para cada compuerta AND. Las salidas de las compuertas AND se aplican a una sola compuerta OR para generar una salida de 1 línea. El tamaño del multiplexor se especifica por el número 2^n de sus líneas de entrada y de la



(a) Diagrama lógico



(c) Diagrama de bloque

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Tabla de función

Figura 5-16 Un multiplexor en línea de 4 a 1

sola línea de salida, implicando así que contiene n líneas de selección. Un multiplexor es a menudo abreviado como MUX.

Como en los decodificadores, los CI multiplexores pueden tener una entrada de activación para controlar la operación de la unidad. Cuando la entrada de activación esté en un estado binario dado, las salidas se inhabilitan o cuando está en el otro estado (el estado de habilitación) el circuito funciona como un multiplexor normal. La entrada de habilitación o activación (algunas veces llamada *strobe*) puede ser usada para expandir dos o más CI multiplexores a un multiplexor digital con un gran número de entradas.

En algunos casos se encapsulan dos o más multiplexores dentro de un CI. Las entradas de selección y activación en los CI de múltiple unidad pueden ser comunes a todos los multiplexores. Como ilustración se muestra en la Figura 5-17* un CI multiplexor cuádruple de 2 líneas a 1 línea. Este tiene cuatro multiplexores cada uno de los cuales puede seleccionar una de dos líneas de entrada. La salida Y_1 puede ser seleccionada para ser igual a A_1 ó B_1. De manera similar, la salida Y_2 podría tener el valor de A_2 ó B_2 y así sucesivamente. Una línea de selección de entrada, S, es suficiente para seleccionar una de dos líneas en todos los cuatro multiplexores. La entrada de control E habilita los multiplexores en el estado 0 y los inhabilita en el estado 1. Aunque el circuito contiene cuatro multiplexores se podría pensar que es un circuito que selecciona una en un par de

* Este es similar al circuito integrado tipo 74157.

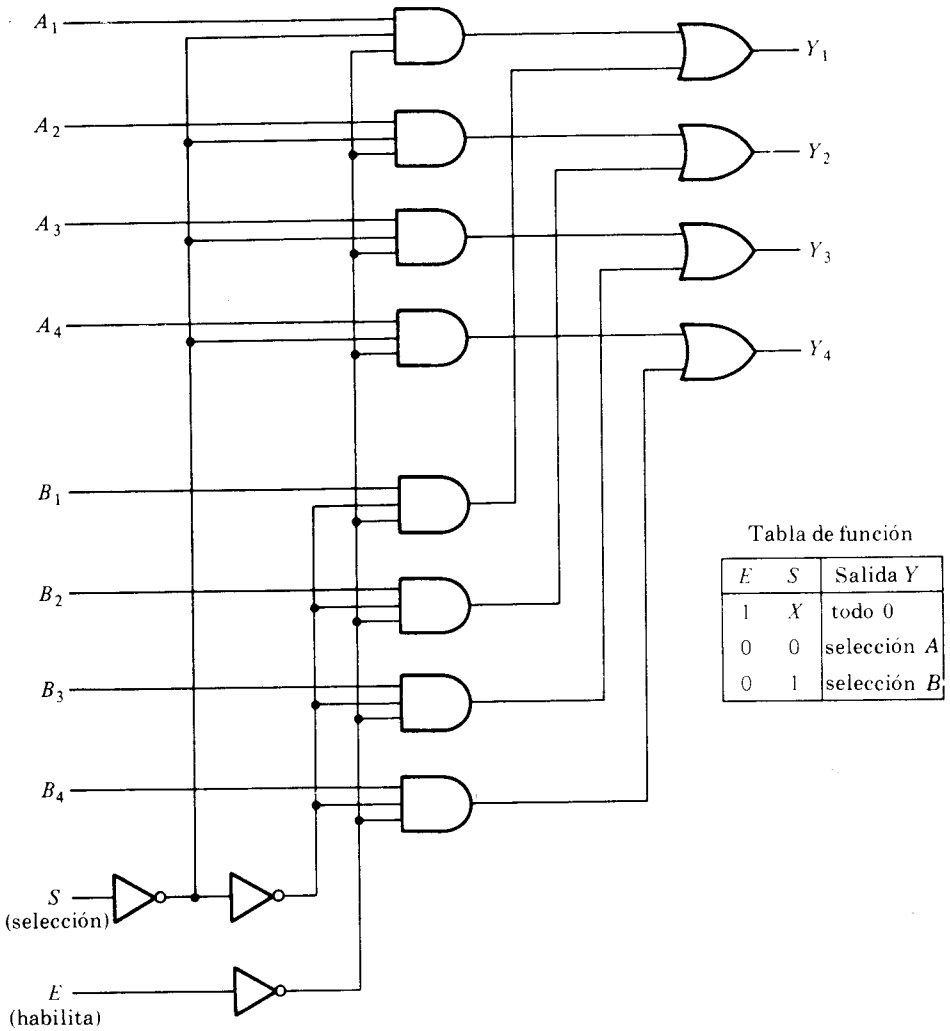


Tabla de función

E	S	Salida Y
1	X	todo 0
0	0	selección A
0	1	selección B

Figura 5-17 Multiplexores cuádruples en línea de 2 a 1

4 líneas de entrada. Como se ve en la tabla de la función, la unidad se selecciona cuando $E = 0$. Entonces, si $S = 0$ las cuatro entradas A tienen una vía hacia las salidas. Por otra parte, si $S = 1$ se seleccionan las otras cuatro entradas B. Las salidas serán todas ceros cuando $E = 1$ sin tener en cuenta el valor de S.

El multiplexor es una función MSI muy útil y tiene una multitud de aplicaciones. Se usa para conectar dos o más fuentes a un solo destino entre las unidades del computador y es útil para construir un sistema de bus común. Estos y otros usos del multiplexor se discutirán en capítulos posteriores conjuntamente con sus aplicaciones particulares. Aquí se demuestran las propiedades generales de este elemento y se muestra cómo puede ser usado para ejecutar una función de Boole.

Ejecución de una función de Boole

Se habría demostrado en la sección anterior que el decodificador puede ser usado para configurar una función de Boole empleando una compuerta OR externa. Un rápido vistazo al multiplexor de la Figura 5-16 revela que es esencialmente un decodificador con una compuerta OR ya disponible. Los términos mínimos fuera del decodificador que va a escogerse pueden controlarse con las líneas de entrada. Los términos mínimos que van a incluirse con la función que se está ejecutando se escogen haciendo sus líneas de entrada correspondientes, iguales a 1 y aquellos términos mínimos no incluidos en la función se inhabilitan al hacer las líneas de entrada iguales a cero. Esto presenta un método para configurar cualquier función de Boole de n variables con un multiplexor de 2^n a 1. Sin embargo, es posible hacer algo mejor que eso.

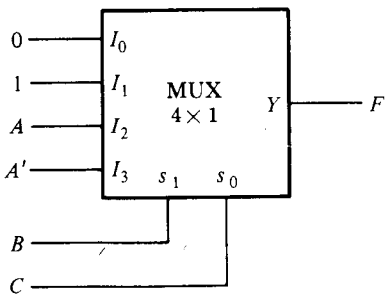
Si se tiene una función de Boole de $n+1$ variables se toman n de estas variables y se conectan a las líneas de selección del multiplexor. La variable restante de la función se usa para las entradas del multiplexor. Si A es esta sola variable, las entradas del multiplexor se escogen para ser A ó A' ó 1 ó 0. Mediante un concienzudo uso de estos cuatro valores para las entradas y conectando las otras variables a las líneas de selección, se puede configurar cualquier función de Boole con un multiplexor. De esta forma es posible generar cualquier función de $n+1$ variables con un multiplexor de 2^n a 1.

Para demostrar este procedimiento con un ejemplo concreto, considérese la función de tres variables:

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$

La función puede ser configurada con un multiplexor de 4 a 1 como se muestra en la Figura 5-18. Dos de las variables B y C se aplican a las líneas de selección en ese orden, es decir, B se conecta a s_1 y C a s_0. Las entradas del multiplexor son 0, 1, A y A'. Cuando $BC = 00$ la salida $F = 0$ ya que $I_0 = 0$. Por tanto, ambos términos mínimos $m_0 = A'B'C'$ y $m_4 = AB'C'$ producen una salida 0, ya que la salida es 0 cuando $BC = 00$ sin tener en cuenta el valor de A. Cuando $BC = 01$, la salida $F = 1$ ya que $I_1 = 1$. Por tanto, ambos términos mínimos $m_1 = A'B'C$ y $m_5 = AB'C$ producen una salida de 1 ya que la salida es 1 cuando $BC = 01$ sin tener en cuenta el valor de A. Cuando $BC = 10$ la entrada I_2 es seleccionada. Como A se conecta a esta entrada, la salida será igual a 1 solamente para el término mínimo $m_6 = ABC'$, pero no para el término mínimo $m_2 = A'BC'$, debido a que $A' = 1$, entonces $A = 0$ y como $I_2 = 0$ se tiene entonces $F = 0$. Finalmente cuando $BC = 11$ se selecciona la entrada I_3. Como A' se conecta a esta entrada, la salida será igual a 1 solamente para el término mínimo $m_3 = A'BC$ pero no para $m_7 = ABC$. Esta información se sumariza en la Figura 5-18(b), la cual es la tabla de verdad de la función que se requiere ejecutar.

La anterior discusión muestra por análisis que el multiplexor configura la función requerida. Se representará ahora un procedimiento general para configurar cualquier función de Boole de n variables con un multiplexor de 2^{n-1} a 1.



(a) Configuración del multiplexor

Término mínimo	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

(b) Tabla de verdad

	I_0	I_1	I_2	I_3
A'	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A'

(c) Tabla de configuración

Figura 5-18 Configurando $F(A, B, C) = \sum (1, 3, 5, 6)$ con un multiplexor

Primero se expresa la función en su forma de suma de términos mínimos. Se asume que la secuencia ordenada de variables escogidas para los términos mínimos es $ABCD$ . . ., donde A es la variable de la extrema izquierda en una secuencia ordenada de n variables y BCD . . . son los $n-1$ variables restantes. Se conectan las $n-1$ variables a las líneas de selección del multiplexor con B conectada a una línea de selección de mayor orden, C a la siguiente línea menor de selección y así sucesivamente hasta la última variable la cual se conecta a la línea de selección de más bajo orden s_0. Considérese la variable A. Como esta variable está en la posición de más alto orden en una secuencia de variables, será complementada en los términos mínimos o hasta $(2^n/2)-1$ los cuales comprenden la primera mitad en la lista de los términos mínimos. La segunda mitad de los términos mínimos tendrán su variable A sin complementar. Para una función de tres variables, A, B, C se tiene ocho términos mínimos. La variable A se complementa en los términos mínimos 0 a 3 y no se complementa en los términos mínimos 4 a 7.

Lístese las entradas del multiplexor y bajo ellas los términos mínimos en dos columnas. La primera fila incluye todos los términos mínimos en los cuales A es complementada y la segunda fila todos los términos mínimos con A no complementada de la manera mostrada en la Figura 5-18(c). Enciérrese en un círculo todos los términos mínimos de la función e inspecciónese cada columna separadamente.

Si los dos términos mínimos en una columna no están en círculo aplíquese 0 a la entrada correspondiente del multiplexor.

Si los dos términos mínimos están en un círculo aplíquese 1 a la entrada correspondiente del multiplexor.

Si el término mínimo inferior está encerrado en un círculo y el superior no lo está aplíquese A a la entrada correspondiente del multiplexor.

Si el término mínimo superior está encerrado en un círculo y el inferior no lo está aplíquese A' a la entrada correspondiente del multiplexor.

Este procedimiento se desprende de las condiciones establecidas durante el análisis previo.

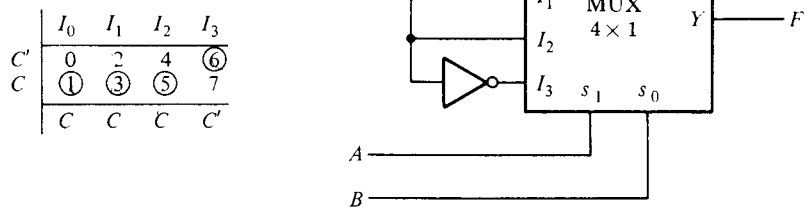
La Figura 5-18(c) muestra la configuración de la función de Boole:

$$F(A, B, C) = \Sigma(1, 3, 5, 6)$$

de la cual se obtiene las conexiones del multiplexor de la Figura 5-18(a). Nótese que B debe conectarse a s_1 y C a s_0.

No es necesario escoger la variable de la extrema izquierda de la secuencia ordenada de una lista de variables para las entradas del multiplexor. De hecho, se pueden escoger cualquiera de las variables para las entradas del multiplexor si se tiene en cuenta la modificación de la tabla de ejecución. Supóngase que se va a configurar la misma función con un multiplexor, pero usando las variables A y B para la línea de selección s_1 y s_0, y la variable C para las entradas del multiplexor. La variable C se complementa en los términos mínimos pares y no se complementa para los impares ya que es la última variable en la secuencia de las variables listadas. El arreglo de las dos filas de términos mínimos en este caso debe ser como se muestra en la Figura 5-19(a). Encerrando en un círculo los términos mínimos y usando las reglas establecidas anteriormente se obtienen las conexiones del multiplexor para la configuración de la función como se ve en la Figura 5-19(b).

En forma similar, es posible usar cualquier variable de la función en las entradas del multiplexor. Se pueden formular varias combinaciones para configurar una función de Boole con multiplexores. De cualquier manera, todas las variables de entrada a excepción de una, se aplican a las líneas de selección. La variable restante o su complemento ó 0 ó 1 se aplican a las entradas del multiplexor.



(a) Tabla de configuración (b) Conexión del multiplexor

Figura 5-19 Configuración alterna para $F(A, B, C) = \sum(1, 3, 5, 6)$

	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
A'	⓪	①	2	③	④	5	6	7
A	⑧	⑨	10	11	12	13	14	⑮
	1	1	0	A'	A'	0	0	A

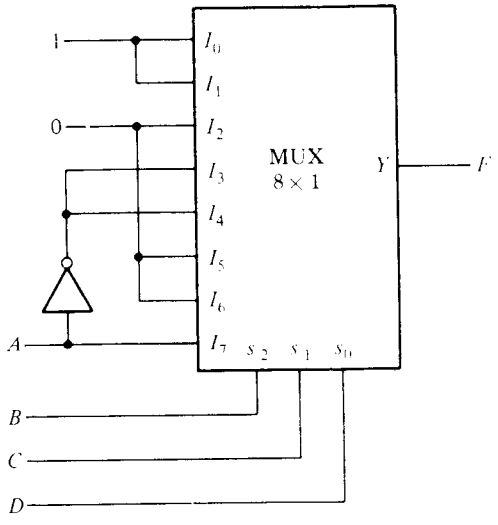


Figura 5-20 Configuración de $F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$

EJEMPLO 5-4: Ejecutar la siguiente función con un multiplexor:

$$F(A, B, C, D) = \Sigma(0, 1, 3, 4, 8, 9, 15)$$

Esta es una función de cuatro variables y por tanto se necesita un multiplexor con tres líneas de selección y ocho entradas. Se escoge aplicar las variables B, C y D a las líneas de selección. La tabla de configuración es la mostrada en la Figura 5-20. La primera mitad de los términos mínimos están asociados con A' y la segunda mitad con A. Encerrando en un círculo los términos mínimos de la función y aplicando las reglas para encontrar los valores para las entradas del multiplexor, se obtiene el circuito mostrado.

Compárese ahora el método del multiplexor con el método del codificador para configurar los circuitos combinacionales. El método del decodificador requiere una compuerta OR para cada función de salida, más sólo se necesita un decodificador para generar todos los términos mínimos. El método del multiplexor usa unidades de menor tamaño pero requiere un multiplexor para cada función de salida. Podría ser razonable asumir que los circuitos combinacionales con una pequeña cantidad de salidas se puedan realizar con multiplexores. Los circuitos combinacionales con muchas funciones de salida probablemente usan menos CI con el método del decodificador.

Aunque los multiplexores y decodificadores se pudieran usar para la ejecución de los circuitos combinacionales, debe tenerse en cuenta que los decodificadores se usan principalmente para decodificar la información binaria y los multiplexores para formar un camino selecto entre múltiples fuentes y un solo destino. Se deberían considerar cuando se diseñan pe-

queños circuitos combinacionales especiales que no se consiguen como funciones MSI. Para los grandes circuitos combinacionales con múltiples entradas y salidas, hay un componente de CI más adecuado y este se presenta en la siguiente sección.

5-7 MEMORIA DE SOLO LECTURA (ROM)

Se vió en la Sección 5-5 que un decodificador genera los 2^n términos mínimos de las n entradas variables. Colocando las compuertas OR para sumar los términos mínimos de las funciones de Boole se podrá generar cualquier circuito combinacional. Una memoria de solo lectura (ROM) que viene de Read Only Memory) es un elemento que incluye el decodificador y las compuertas OR dentro de una sola cápsula de CI. Las conexiones entre las salidas del decodificador y las entradas de las compuertas OR pueden especificarse para cada configuración particular "programando" la ROM. La ROM se usa a menudo para configurar un circuito combinacional complejo en una cápsula de CI y así eliminar los cables de conexión.

Una ROM es esencialmente un dispositivo (o acumulador) de memoria en el cual se almacena un conjunto fijo de información binaria. La información binaria debe especificarse por el usuario y luego enclavarse en la unidad para formar el patrón de interconexión requerida. Las ROM vienen con enlaces internos especiales que pueden estar fusionados o abiertos. La interconexión deseada para una aplicación particular requiere que ciertos enlaces estén fusionados para formar los caminos del circuito necesarios. Una vez que se establezca un patrón para una ROM, este permanecerá fijo aunque se haga un corte de corriente y luego se restablezca.

Un diagrama de bloque de una ROM se muestra en la Figura 5-21. Este consiste en n líneas de entrada y m líneas de salida. Cada combinación de bits de las variables de entrada se llama una *dirección*. Cada combinación de bits que sale por las líneas de salida se llama una *palabra*. El número de bits por palabra es igual al número de líneas de salida m. Una dirección es esencialmente un número binario que denota uno de los términos mínimos de n variables. El número de direcciones diferentes posibles con n variables de entrada es 2^n. Una palabra de salida puede ser seleccionada por una dirección única y como hay 2^n direcciones diferentes

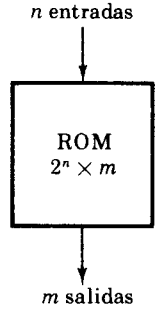


Figura 5-21 Diagrama de bloque de una ROM

en una ROM, hay 2^n palabras diferentes qué se dice que están acumuladas en la unidad. La palabra disponible en las líneas de salida, en cualquier momento dado, depende del valor de la dirección aplicada a las líneas de entrada. Una ROM se caracteriza por el número de palabras 2^n y el número de bits por palabra m. Esta terminología se usa debido a la similitud entre la memoria de solo lectura y la memoria de lectura-escritura que se presenta en la Sección 7-7.

Considérese una ROM de 32×8. La unidad consiste en 32 palabras de 8 bits cada una. Esto significa que hay ocho líneas de salida y 32 palabras distintas almacenadas en la unidad, cada una de las cuales puede aplicarse a las líneas de salida. La palabra particular seleccionada que está presente en las líneas de salida se determinan a partir de las cinco líneas de entrada. Hay solamente cinco entradas en una ROM de 32×8 porque $2^5 = 32$ y con cinco variables se puede especificar 32 direcciones o términos mínimos. Para cada dirección de entrada hay una palabra única seleccionada. Así, si una dirección de entrada es 00000, se selecciona la palabra número 0 y esta aparece en las líneas de salida. Si la dirección de entrada es 11111, se selecciona la palabra número 31 y se aplica a las líneas de salida. Entre la primera y la última hay otras 30 direcciones que pueden seleccionar otras 30 palabras.

El número de palabras direccionadas en una ROM se determina del hecho de que se necesitan n líneas de entrada para especificar 2^n palabras. Una ROM se especifica algunas veces por el número total de bits que contiene, el cual será $2^n \times m$. Por ejemplo, una ROM de 2048 bits puede organizarse como 512 palabras de 4 bits cada una. Esto significa que la

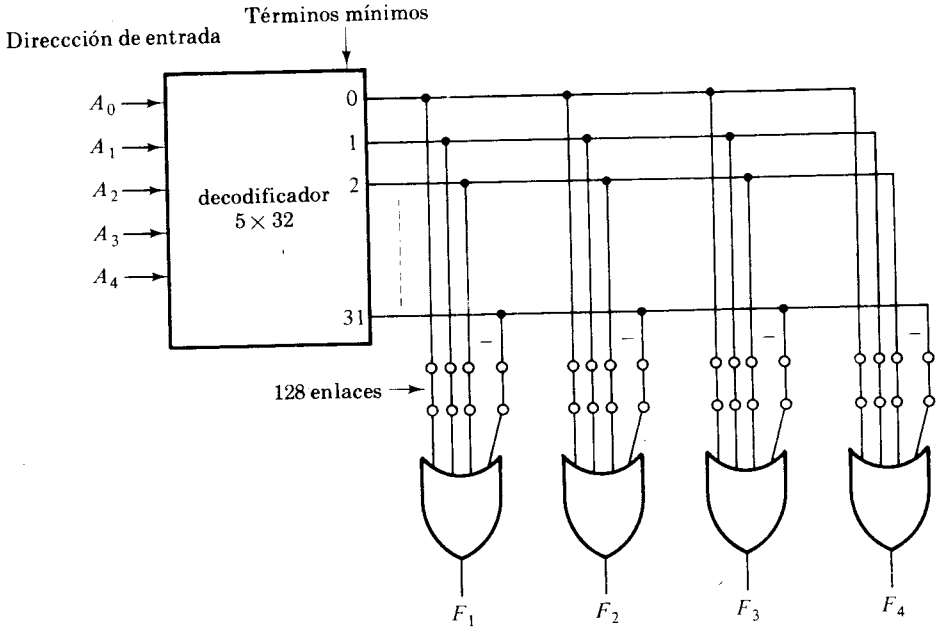


Figura 5-22 Construcción lógica de una ROM de 32×4

unidad tiene 4 líneas de salida y 9 líneas de entrada para especificar $2^9 = 512$ palabras. El número total de bits en la unidad es $512 \times 4 = 2.048$.

Internamente, la ROM es un circuito combinacional con compuertas AND conectadas como decodificador y un número de compuertas OR igual al número de salidas de la unidad. La Figura 5-22 muestra una construcción lógica interna de una ROM de 32×4. Las cinco variables de entrada se decodifican en 32 líneas por medio de 32 compuertas AND y 5 inversores. Cada salida del decodificador representa uno de los términos mínimos de una función de cinco variables. Cada una de las 32 direcciones selecciona una y sólo una salida del decodificador. La dirección es un número de 5 bits aplicado a las entradas y el término mínimo seleccionado por fuera del decodificador es el marcado con el número decimal equivalente. Las 32 salidas del decodificador están conectadas por medio de *enlaces* a cada compuerta OR. Solamente cuatro de estos enlaces se muestran en el diagrama pero realmente cada compuerta OR tiene 32 entradas y cada entrada pasa a través de un enlace que puede estar cortado si así se desea.

La ROM es una configuración de dos niveles en forma de suma de términos mínimos. No tiene que ser una configuración AND-OR, pero puede ser cualquier otra posible configuración de términos mínimos de dos niveles. El segundo nivel es normalmente una conexión de lógica cableada (ver Sección 3-7) para facilitar la función de los enlaces.

Las ROM tienen muchas aplicaciones importantes en el diseño de sistemas de computadores digitales. Su uso para la configuración de circuitos combinacionales complejos es justamente una de esas aplicaciones. Otros usos de las ROM se presentan en otras partes del libro conjuntamente con aplicaciones particulares.

Configuración de lógica combinacional

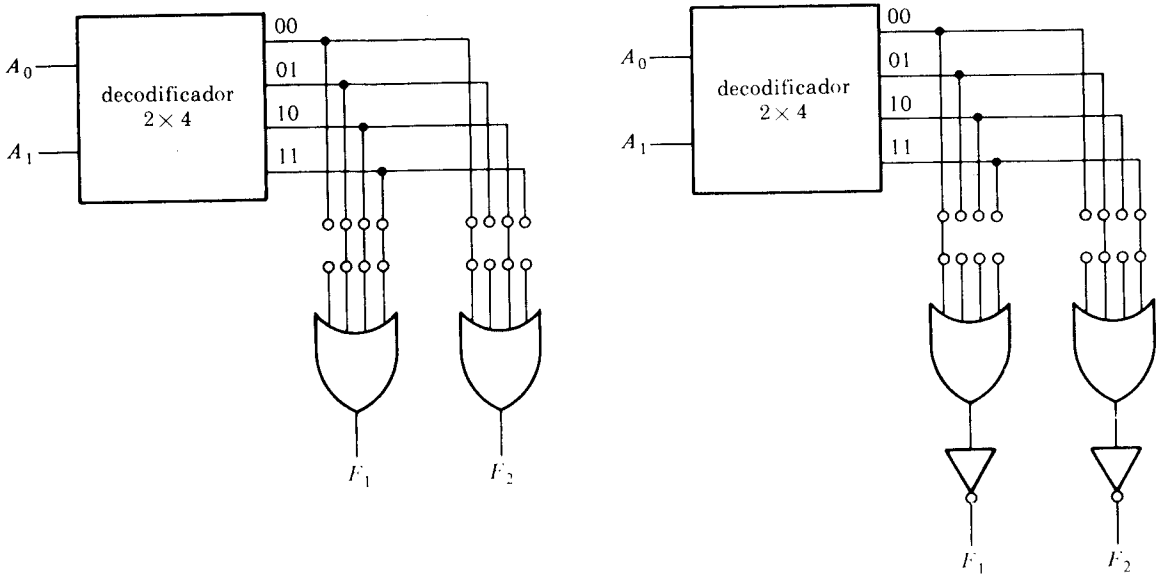
Del diagrama lógico de la ROM, es claro que cada salida produce la suma de todos los términos mínimos de n variables de entrada. Recuérdese que una función de Boole puede ser expresada en forma de suma de términos mínimos. Al romper los enlaces de aquellos términos mínimos que no se incluyen en la función, cada salida de la ROM puede hacer representar la función de Boole de una de las variables de salida en un circuito combinacional. Para un circuito combinacional de n entradas y m salidas se necesita una ROM de $2^n \times m$. La ruptura de los enlaces se refiere a la *programación* de la ROM. El diseñador necesita solamente especificar una tabla del programa ROM que da la información para los caminos necesarios en la ROM. La programación actual es un procedimiento del material (hardware) que sigue las especificaciones listadas en la tabla de programación.

Para aclarar el proceso es necesario un ejemplo específico. La tabla de verdad en la Figura 5-23(a) especifica un circuito combinacional con dos entradas y dos salidas. Las funciones de Boole pueden expresarse en suma de términos mínimos:

$$F_1(A_1, A_0) = \Sigma(1, 2, 3)$$
$$F_2(A_1, A_0) = \Sigma(0, 2)$$

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

(a) Tabla de verdad



(b) ROM con compuertas AND-OR

(c) ROM con compuertas AND-OR-invertido

Figura 5-23 Configuración del circuito combinacional con una ROM de 4×2

Cuando se configura un circuito combinacional por medio de una ROM, las funciones deben expresarse en suma de términos mínimos o mejor aún por una tabla de verdad. Si la salida de las funciones se simplifica, se encuentra que el circuito necesita solamente una compuerta OR y un inversor. Obviamente, este es un circuito combinacional simple para ser ejecutado con una ROM. La ventaja de las ROM es su uso en circuitos combinacionales complejos. Este ejemplo solamente demuestra el procedimiento y no debe considerarse en una situación práctica.

La ROM que configura el circuito combinacional debe tener dos entradas y dos salidas de tal manera que su tamaño deberá ser 4×2. La Figura 5-23(b) muestra la construcción interna de una ROM. Es necesario determinar cuál de los ocho enlaces disponibles deben romperse y cuáles deben dejarse sin tocar. Esto puede hacerse fácilmente de las funciones de salida listadas en la tabla de verdad. Aquellos términos mínimos que especifican una salida de 0 no deben tener un camino a la salida a través de una compuerta OR. Así, pera este caso particular la tabla de verdad muestra tres ceros y sus correspondientes enlaces con las compuertas OR que deben quitarse. Es obvio que se debe asumir que un circuito abierto a una compuerta OR se comporta como una entrada de 0.

Algunas ROM vienen con un inversor después de cada una de las compuertas OR y como consecuencia se especifica que inicialmente tienen todos 0 en sus entradas. El procedimiento de programación de tales ROM requiere que se abran los enlaces de los términos mínimos (o direcciones) que especifiquen una salida de 1 en la tabla de verdad. La salida de la compuerta OR complementa la función una vez más para producir una salida normal. Esto se muestra en la ROM de la Figura 5-23(c).

El ejemplo anterior demuestra el procedimiento general para ejecutar un circuito combinacional con una ROM. A partir del número de entradas y salidas en el circuito combinacional, se determina primero el tamaño de la ROM requerido. Luego se obtiene la tabla de verdad de programación de la ROM; no se necesita ninguna otra manipulación o simplificeación. Los ceros (o unos) en las funciones de salida de la tabla de verdad especifican directamente aquellos enlaces que deben ser removidos para producir el circuito combinacional requerido en la forma de suma de términos mínimos.

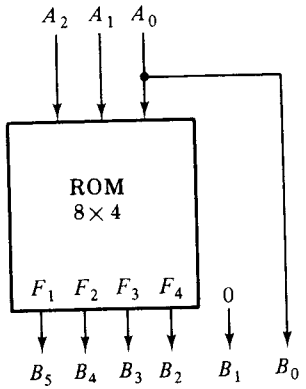
En la práctica, cuando se diseña un circuito por medio de una ROM, no es necesario mostrar enlaces de las conexiones de las compuertas internas dentro de la unidad como se hizo en la Figura 5-23; lo cual fue mostrado para propósitos de demostración solamente. Todo lo que el diseñador tiene que hacer es especificar la ROM (o su número asignado) y dar la tabla de verdad de la ROM como en la Figura 5-23(a). La tabla de verdad da toda la información para programar la ROM. No se necesita un diagrama interno que acompañe la tabla de verdad.

EJEMPLO 5-5: Diseñar un circuito combinacional usando una ROM. El circuito acepta un número de 3 bits y genera un número binario de salida igual al cuadrado del número de entrada.

El primer paso es deducir la tabla de verdad para el circuito combinacional. En la mayoría de los casos es todo lo que se nece-

Tabla 5-5 Tabla de verdad para el circuito del Ejemplo 5-5

Entradas			Salidas						Decimal
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



(a) Diagrama de bloque

A_2	A_1	A_0	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) Tabla de verdad de la ROM

Figura 5-24 Configuración de la ROM del Ejemplo 5-5

sita. En algunos casos se puede encajar una tabla de verdad más pequeña para la ROM usando ciertas propiedades en la tabla de verdad del circuito combinacional. La Tabla 5-5 es la tabla de verdad para el circuito combinacional. Se necesitan las tres entradas y las tres salidas para acomodar todos los números posibles. Se nota que la salida B_0 es siempre igual a la entrada A_0 de tal manera que no es necesario generar B_0 con la ROM ya que es igual a una variable de entrada. Sin embargo, la salida B_1 es siempre 0, de tal manera que siempre es conocida. Se necesita generar solamente cuatro entradas con una ROM; las otras dos se obtienen fácilmente. El tamaño mínimo de la ROM debe tener tres entradas y cuatro salidas. Las tres entradas especifican ocho palabras de tal manera que el tamaño de la ROM debe ser 8×4. La configuración con ROM se muestra en la Figura 5-24. Las tres entradas especifican ocho palabras con cuatro bits cada una. Las otras dos salidas de los circuitos combinacionales son iguales a

0 y A_0. La tabla de verdad de la Figura 5-24 especifica toda la información necesaria para programar la ROM y el diagrama de bloque muestra las conexiones requeridas.

Tipos de ROM

Los caminos necesarios en una ROM pueden ser programados de dos maneras diferentes. La primera se llama *programación por máscara* y la hace el fabricante durante el último proceso de fabricación de la unidad. El procedimiento para fabricar una ROM requiere que el cliente llene la tabla de verdad según lo que se desea que la ROM satisfaga. La tabla de verdad debe ser entregada en una forma especial suministrada por el fabricante. Muy a menudo, se entrega en cinta de papel o tarjetas perforadas en el formato especificado en la hoja de datos de una ROM particular. El fabricante hace la máscara correspondiente para que los caminos produzcan unos y ceros de acuerdo a la tabla de verdad del cliente. Este procedimiento es muy costoso ya que el vendedor le carga al cliente una tarifa especial por hacerle una ROM con máscara. Por esta razón, la programación con máscara es económica solamente si se van a fabricar grandes cantidades del mismo tipo de configuración de ROM.

Para pequeñas cantidades, es más económico usar un segundo tipo de ROM llamado *memoria programable* de solo lectura o PROM (de programable read-only memory). Cuando se ordenan, las unidades PROM contienen ceros (o unos) en cada bit de las palabras almacenadas. Los enlaces en el PROM se rompen por medio de pulsos de corriente a través de los terminales de salida. Un enlace roto define un estado binario y uno no roto representa el otro estado. Esto le permite al usuario programar la unidad en su propio laboratorio para lograr la relación deseada entre las direcciones de entrada y las palabras almacenadas. Comercialmente se obtienen unidades especiales llamadas *programadores de PROM* para facilitar este procedimiento. De todas formas, todos los procedimientos para programar las ROM son procedimientos de los *materiales* (hardware) aunque se use la palabra *programación*.

El procedimiento de los materiales para programar ROM o PROM es irreversible y una vez programados el patrón dado es permanente y no puede alterarse. Una vez que se ha establecido un patrón de bits se debe descartar la unidad si se quiere cambiar el patrón de bits. Un tercer tipo de unidad es la llamada *PROM borrable* o EPROM (de erasable PROM). Las EPROM pueden ser recuperadas a su valor inicial (todos unos o todos ceros) aunque se hayan cambiado previamente. Cuando una EPROM se coloca bajo una luz ultravioleta especial por un periodo dado de tiempo, la radiación de onda corta descarga los puentes internos que sirven de contactos. Una vez borrada la ROM regresa a su estado inicial para ser reprogramada. Ciertas ROM pueden ser borradas con señales eléctricas en vez de luz ultravioleta y se les llama algunas veces *ROM eléctricamente alterable* o EAROM.

La función de una ROM puede interpretarse de dos maneras diferentes. La primera interpretación es la de una unidad que configura cualquier circuito combinacional. Desde este punto de vista, cada terminal de salida

se considera separadamente como una salida de una función de Boole expresada en suma de términos mínimos. La segunda interpretación considera la ROM como una unidad de almacenamiento que tiene un patrón fijo de cadenas de bits llamadas *palabras*. Visto de esta forma, las entradas especifican una *dirección* para una palabra específica almacenada que se aplica luego a las salidas. Por ejemplo, la ROM de la Figura 5-24 tiene tres líneas de dirección las cuales especifican ocho palabras acumuladas de la manera dada en la tabla de verdad. Cada palabra tiene cuatro bits de longitud. Esta es la razón por la cual se le ha dado a la unidad el nombre de *memoria de solo lectura. Memoria* se usa comúnmente para designar una unidad de almacenamiento. *Lectura* se usa para implicar que el contenido de una palabra especificada por una dirección en una unidad de almacenamiento se localiza en los terminales de salida. Así, una ROM es una unidad de memoria con un patrón fijo de palabra que puede ser leído bajo la aplicación de una dirección dada. El patrón de bits en la ROM es permanente y no puede cambiarse durante la operación normal.

Las ROM se usan extensamente para ejecutar circuitos combinacionales complejos directamente de sus tablas de verdad. Son muy útiles para convertir de un código binario a otro (tal como ASCII a EBCDIC o viceversa), para funciones aritméticas como multiplicadores, para mostrar caracteres en un tubo de rayos catódicos, y en cualquier otra aplicación que requiera un gran número de entradas y salidas. Se emplean también en el diseño de unidades de control de los sistemas digitales. Como tales, se usan para almacenar patrones fijos de bits que representen una secuencia de variables de control necesarios para habilitar las diferentes operaciones en el sistema. Una unidad de control que utiliza una ROM para almacenar información de control binario se llama *una unidad de control microprogramada.* El Capítulo 10 tratará este tema en más detalles.

5-8 ARREGLO LOGICO PROGRAMABLE (PLA)

Un circuito combinacional puede tener ocasionalmente condiciones de no importa. Cuando se configura con una ROM una condición de no importa se convierte en una dirección de entrada que nunca ocurre. Las palabras en las direcciones de no importa no necesitan ser programadas y pueden dejarse en su estado original (todos ceros o todos unos). El resultado es que no todos los patrones de bits disponibles en la ROM se usan, lo cual se considera como un desperdicio de equipo disponible.

Considérese por ejemplo, un circuito combinacional que convierte un código de tarjeta de 12 bits a un código alfanumérico interno de 6 bits, como se lista en la Tabla 1-5. El código de tarjeta de entradas consiste en 12 líneas designadas por 0, 1, 2, . . . , 9, 11, 12. El tamaño de la ROM para configurar el conversor de código debe ser 4096×6, ya que hay 12 entradas y 6 salidas. Hay solamente 47 entradas válidas para el código de tarjeta y el resto de combinaciones son condiciones de no importa. Se usan así solamente 47 palabras de las 4096 disponibles. Las 4049 palabras restantes no se usan y se desperdician.

Para aquellos casos en los cuales el número de condiciones de no importa es excesivo, es más económico usar un segundo tipo de componente

LSI llamado *arreglo lógico programable* o PLA (viene de *programable logic array*). Un PLA es similar a una ROM en concepto; sin embargo el PLA no produce la decodificación completa de las variables y no genera todos los términos mínimos como en una ROM. En un PLA, el decodificador se remplaza mediante un grupo de compuertas AND, cada una de las cuales pueden ser programadas para generar un término producto de las variables de entrada. Las compuertas AND y OR dentro del PLA se fabrican inicialmente con enlaces entre ellas. Las funciones específicas de Boole se ejecutan en la forma de suma de productos al abrir los enlaces adecuados y dejar las conexiones deseadas.

Un diagrama de bloque de un PLA se muestra en la Figura 5-25. Este consiste en n entradas, m salidas, k términos de producto y m términos de suma. Los términos de producto constituyen un grupo de k compuertas AND y los términos de suma constituyen un grupo de m compuertas OR. Los enlaces se colocan entre todas las entradas n y sus valores complementados. Otro grupo de enlaces en los inversores de salida permite que se genere la función de salida o en la forma de AND-OR o en la forma AND-OR invertida. Con el enlace del inversor en su lugar, se puentea el inversor dando una configuración AND-OR. Cuando se rompe el enlace el inversor se vuelbe parte del circuito y la función se configura en la forma AND-OR invertida.

El tamaño del PLA se especifica por el número de entradas, el número de términos de producto y el número de salidas (el número de términos de suma es igual al número de salidas). Un típico PLA tiene 16 entradas, 48 términos producto y 8 salidas.* El número de enlaces programados es $2n \times k + k \times m + m$ mientras que los de la ROM son $2^n \times m$.

La Figura 5-26 muestra una construcción interna de un PLA específico. Tiene tres entradas, tres términos producto y dos salidas. Tal PLA es muy pequeño para encontrarse comercialmente; se presenta aquí solamente para propósito de demostración. Cada entrada y su complemento se conecta por medio de enlaces a las entradas de todas las compuertas AND. Las salidas de las compuertas AND se conectan por medio de enlaces a cada entrada de las compuertas OR. Se suministran dos enlaces más con los inversores de salida. Al romper los enlaces seleccionados y dejar otros en lugar, es posible ejecutar configuraciones de funciones de Boole en la forma de suma de productos.

De la misma forma que la ROM, el PLA puede ser programable por máscara o programable por el usuario (programación de campo). Con un

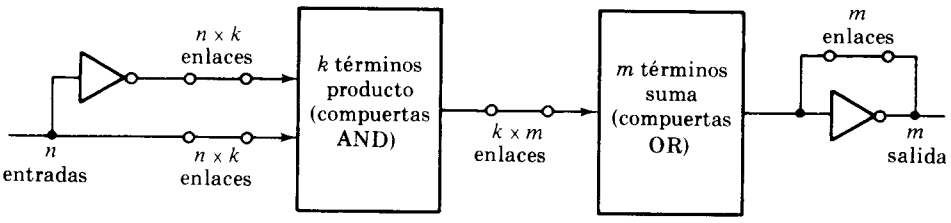


Figura 5-25 Diagrama de bloque del PLA

*El CI TTL tipo 82S100.

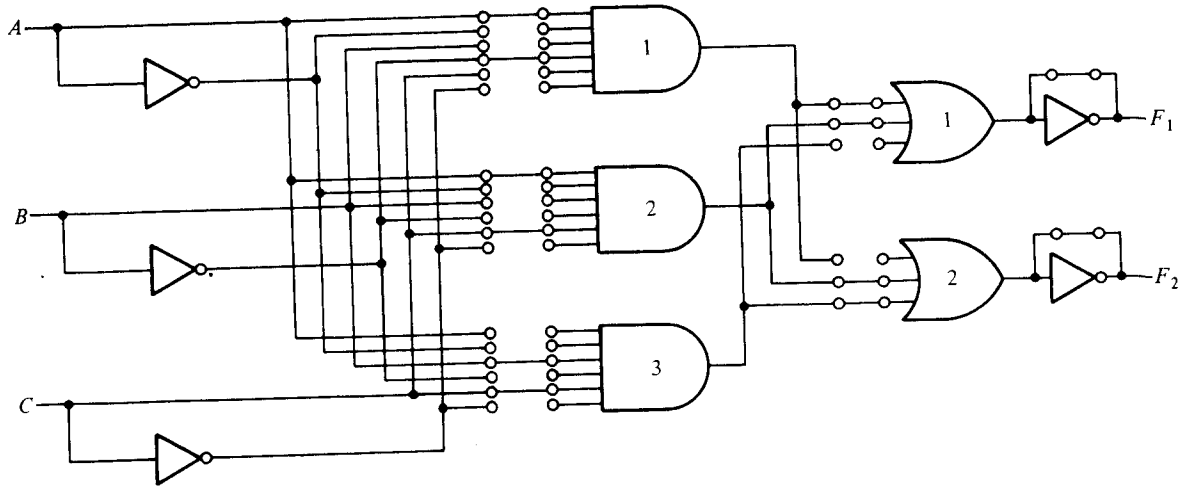


Figura 5-26 PLA con 3 entradas, 3 términos producto y 2 salidas; este configura el circuito combinacional especificado en la Figura 5-27

PLA programable por máscara, el cliente debe entregar una tabla de programación del PLA al fabricante. Esta tabla se usa por el fabricante para producir un PLA hecho para el cliente con los caminos internos requeridos entre las entradas y las salidas. Un segundo tipo de PLA disponible se llama *arreglo lógico programable en el campo* o FPLA (de field programmable logic array). El FPLA puede ser programado por el usuario por medio de ciertos procedimientos recomendados. Hay programadores comerciales de materiales (hardware) para usar conjuntamente con ciertos FPLA.

Tabla de programa del PLA

El uso de un PLA debe ser considerado para los circuitos combinacionales que tienen un gran número de entradas y salidas. Es superior a una ROM para circuitos que tienen un gran número de condiciones de no importa. El ejemplo presentado a continuación demuestra cómo se programa un PLA. Manténgase en mente cuando se observe el ejemplo que tal circuito sencillo no necesita un PLA ya que su configuración puede ejecutarse más económicamente con compuertas SSI.

Considérese la tabla de verdad del circuito combinacional mostrado en la Figura 5-27(a). Aunque una ROM configure las funciones en la forma de suma de términos mínimos un PLA configura las funciones en la forma de suma de productos. Cada término de producto en la expresión requiere una compuerta AND. Como el número de compuertas AND en un PLA es finito, es necesario simplificar la función a un número mínimo de términos de producto para poder minimizar el número de compuertas AND usadas. Las funciones simplificadas en suma de productos se obtienen de los mapas de la Figura 5-27(b).

$$F_1 = AB' + AC$$

$$F_2 = AC + BC$$

Hay tres términos de producto distintos en este circuito combinacional: AB', AC y BC. El circuito tiene tres entradas y dos salidas; así el PLA de la Figura 5-26 puede usarse para configurar este circuito combinacional.

La programación del PLA significa que se especifican los caminos en su patrón AND-OR-NOT. Una tabla de programa de PLA típica se muestra en la Figura 5-27(c). Esta consiste en tres columnas. La primera columna lista los términos de producto numéricamente. La segunda columna especifica los caminos necesarios entre las entradas y las compuertas AND. La tercera columna especifica los caminos entre las compuertas AND y las OR. Bajo cada variable de salida, se escribe una V (verdadero) si la función debe complementarse con el inversor de salida. Los términos de Boole listados a la izquierda no son parte de la tabla; ellos se incluyen solamente como referencia.

Para cada término producto, se marcan las entradas con 1, 0, ó - (guión). Si la variable en el término producto aparece en su forma normal (no tildada), la variable de entrada correspondiente se marca con un 1. Si aparece complementada (tildada) se marca con un 0. Si la variable está ausente

en el término producto se marca con un guión. Cada término producto se asocia con una compuerta AND. Los caminos entre las entradas y las compuertas AND se especifican bajo la columna llamada *entradas*. Un 1 en la columna de entrada especifica un camino desde la correspondiente entrada a la entrada de la compuerta AND que forma el término producto. Un 0 en la columna de entrada especifica un camino entre la entrada correspondiente complementada y la entrada de la compuerta AND. Un guión no especifica conexión. Los enlaces adecuados se rompen y los que quedan forman los caminos deseados como se muestra en la Figura 5-26. Se asume que los terminales abiertos en la compuerta AND se comportan como una entrada de 1.

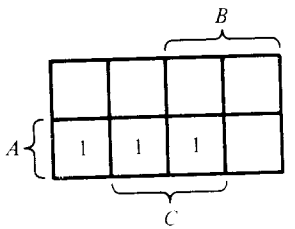
Los caminos entre las compuertas AND y OR se especifican bajo las columnas llamadas *salidas*. Las variables de salida se marcan con unos para aquellos términos producto que formulan la función. En el ejemplo de la Figura 5-27 se tiene:

$$F_1 = AB' + AC$$

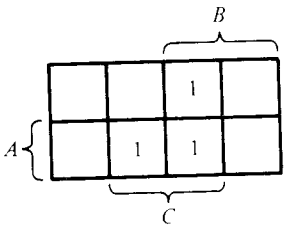
de tal forma que F_1 se marca con un 1 para los términos producto 1 y 2 y con un guión para el término producto 3. Cada término producto que tiene

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

(a) Tabla de verdad



$F_1 = AB' + AC$



$F_2 = AC + BC$

(b) Simplificación por mapa

	Término producto	Entradas A	B	C	Salidas F_1	F_2
AB'	1	1	0	–	1	
AC	2	1	–	1	1	1
BC	3	–	1	1	–	1
					T	T	T/C

(c) Tabla de programa del PLA

Figura 5-27 Pasos necesarios en la configuración del PLA

un 1 en la columna de salida requiere un camino desde la compuerta AND correspondiente hasta la compuerta de salida OR. Aquellos marcados con un guión no especifican conexión. Finalmente una salida V (verdadera) indica que el enlace a través del inversor de salida permanece en su lugar y un C (complemento) indica que el enlace correspondiente está roto. Los caminos internos del PLA para este circuito se muestran en la Figura 5-26. Se asume que un terminal abierto en una compuerta OR se comporta como un 0 y que un corto circuito a través del inversor de salida no daña el circuito.

Cuando se diseña un sistema digital con un PLA no es necesario mostrar las conexiones de la unidad como fue hecho en la Figura 5-26. Todo lo que se necesita es una tabla de programación del PLA mediante la cual se puede programar el PLA para dar los caminos adecuados.

Cuando se configura un circuito combinacional con PLA, se debe hacer una investigación cuidadosa para poder reducir el número total de términos producto ya que un PLA podría tener un número finito de términos AND. Esto puede hacerse simplificando cada función al mínimo número de términos. El número de literales en un término no es importante ya que se tienen disponibles todas las variables de entrada. Los valores verdaderos y de complemento de la función deben simplificarse para ver cual se puede expresar con menos términos producto y cual produce términos producto que son comunes a otras funciones.

EJEMPLO 5-6: Un circuito combinacional se define por las funciones:

$$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$$
$$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$$

Configúrese el circuito con un PLA de 3 entradas cuatro términos producto y dos salidas.

Las dos funciones se multiplican en los mapas de la Figura 5-28. Ambos valores verdaderos y complementos de la función se simplifican. Las combinaciones que dan un número mínimo de términos producto son:

$$F_1 = (B'C' + A'C' + A'B')'$$
$$F_2 = B'C' + A'C' + ABC$$

Esto produce solamente cuatro términos producto diferentes: $B'C', A'C', A'B'$ y ABC. La tabla programa del PLA para esta combinación se muestra en la Figura 5-28. Nótese que la salida F_1 es la salida normal (verdadera) aunque se marque una C bajo ella. Esto es debido a que F_1' se genera *antes* del inversor de salida. El inversor complementa la función para producir F_1 a la salida.

El circuito combinacional para este ejemplo es muy pequeño para una configuración práctica con un PLA. Este se ha presentado aquí solamente

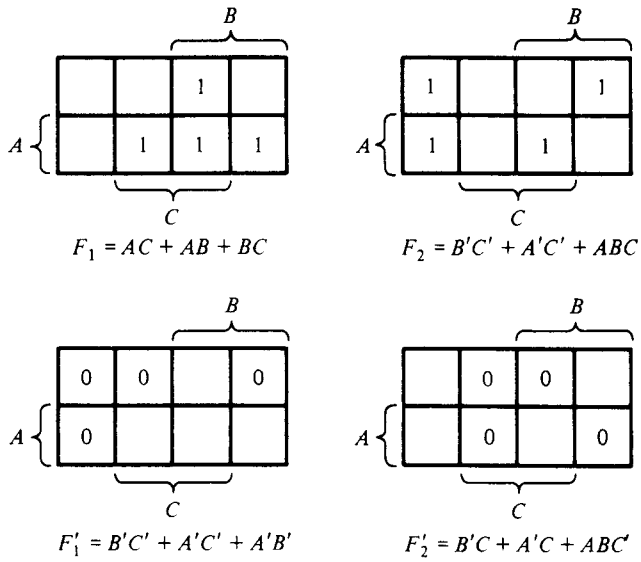


Tabla de programa de un PLA

	Términos productos	Entradas A	B	C	Salidas F_1	F_2	
$B'C'$	1	–	0	0	1	1	
$A'C'$	2	0	–	0	1	1	
$A'B'$	3	0	0	–	1	–	
ABC	4	1	1	1	–	1	
					C	T	T/C

Figura 5-28 Solución del Ejemplo 5-6

para propósitos de demostración. Un PLA típico comercial tiene más de 10 entradas y cerca de 50 términos producto. La simplificación de las funciones de Boole con tantas variables debe llevarse a cabo por medio del método de tabulado u otro método de simplificación a base de computador. Aquí es donde el programa de computador puede ayudar al diseño complejo de los sistemas digitales. El programa del computador debe simplificar cada función del circuito combinacional y su complemento al mínimo número de términos. El programa selecciona el número mínimo de términos diferentes para cubrir todas las funciones en su forma verdadera o de complemento.

5-9 NOTAS CONCLUYENTES

Este capítulo presenta una variedad de métodos de diseño para los circuitos combinacionales. También presenta y explica un número de circuitos MSI y LSI que pueden ser usados para diseñar sistemas digitales más

complicados. El énfasis aquí fue sobre la lógica combinacional MSI y las funciones LSI. Las funciones de la lógica secuencial MSI se discutirán en el Capítulo 7. El procesador y control MSI y las funciones LSI se presentarán en los Capítulos 9 y 10. Los componentes del microcomputador LSI se introducirán en el Capítulo 12.

Las funciones MSI presentadas aquí y otras disponibles comercialmente se describen en los libros de especificaciones o catálogos. Los libros de CI contienen descripciones exactas de muchos MSI y otros circuitos integrados. Algunos de estos libros de datos se listan en las referencias que se darán más adelante.

Los circuitos MSI y LSI pueden usarse en una variedad de aplicaciones. Algunas de estas aplicaciones fueron discutidas a lo largo de este capítulo, algunas fueron incluidas en problemas y otros serán encontradas en capítulos siguientes conjuntamente con sus aplicaciones particulares. Los diseñadores recursivos pueden encontrar muchas otras aplicaciones que se ajusten a sus necesidades particulares. Los fabricantes de circuitos integrados publican numerosas *notas de aplicación* que sugieren la utilización posible de sus productos. Una lista de notas de aplicación puede obtenerse escribiendo a los fabricantes directamente o solicitándola directamente a sus representantes locales.

REFERENCIAS

1. Mano, M. M., *Computer System Architecture.* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1976.
2. Morris, R. L., y J. R. Miller, eds., *Designing with TTL Integrated Circuits.* Nueva York: McGraw-Hill Book Co., 1971.
3. Blakeslee, T. R., *Digital Design with Standard MSI and LSI.* Nueva York: John Wiley & Sons, 1975.
4. Barna A., y D. I. Porat, *Integrated Circuits in Digital Electronics.* Nueva York: John Wiley & Sons, 1973.
5. Lee, S. C., *Digital Circuits and Logic Design,* Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1976.
6. Semiconductor Manufacturers Data Books (Consultar la última edición):
   (a) *The TTL Data Book for Design Engineers.* Dallas, Texas: Texas Instruments, Inc.
   (b) *The Fairchild Semiconductor TTL Data Book.* Mountain View, Calif.: Fairchild Semiconductor.
   (c) *Digital Integrated Circuits.* Santa Clara, Calif.: National Semiconductor Corp.
   (d) *Signetics Digital, Linear, MOS.* Sunnyvale, Calif.: Signetics.
   (e) *MECL Integrated Circuits Data Book.* Phoenix, Ariz.: Motorola Semiconductor Products, Inc.
   (f) *RCA Solid State Data Book Series.* Somerville, N. J.: RCA Solid State Div.

PROBLEMAS

5-1. Diseñe un convertidor de código de exceso 3 a BDC usando un circuito MSI de sumadores completos de 4 bits.

5-2. Usando cuatro circuitos MSI, construya un sumador paralelo binario para sumar dos números binarios de 16 bits. Marque todos los arrastres entre los circuitos MSI.

5-3. Usando 4 compuertas OR-exclusivas y un circuito MSI de sumadores completos de 4 bits, construya un sumador sustractor paralelo. Use una variable de selección de entrada V de tal manera que cuando $V = 0$, el circuito suma y cuando $V = 1$, el circuito resta. (*Sugerencia:* use la sustracción por complemento de 2.)

5-4. Deduzca la ecuación de dos niveles para el bit de arrastre de salida C_5 mostrado en el generador de bit de arrastre posterior de la Figura 5-5.

5-5. (a) Usando el procedimiento de configuración AND-OR invertida descrito en la Sección 3-7, demuestre que el bit de arrastre de salida en el sumador completo puede expresarse como:

$$C_{i+1} = G_i + P_iC_i = (G_i'P_i' + G_i'C_i')'$$

(b) El CI tipo 74182 es un circuito MSI generador de bit de arrastre posterior que genera los bit de arrastre conjuntamente con las compuertas AND-OR invertida. El circuito MSI asume que los terminales de entrada tienen los complementos de G, P y de C_1. Deduzca las funciones de Boole para los bits de arrastre posteriores C_2, C_3 y C_4 en este CI. (*Sugerencia:* use el método de ecuación sustitución para derivar los arrastres en términos de C_1')

5-6. (a) Redefina la programación y generación de los arrastres de la siguiente forma:

$$P_i = A_i + B_i$$
$$G_i = A_iB_i$$

Demuestre que el arrastre de salida y la suma de salida de un sumador completo se convierte en:

$$C_{i+1} = (C_i'G_i' + P_i')' = G_i + P_iC_i$$
$$S_i = (P_iG_i') \oplus C_i$$

(b) El diagrama lógico del primer estado del sumador en paralelo de 4 bits como se configura en el CI tipo 74283 y se muestra en la Figura P5-6. Identifique los terminales P_i' y G_i' como se definieron en (a) y demuestre que el circuito puede configurar un sumador completo.

(c) Obtenga los arrastres de salida C_3 y C_4 en función de P_1', P_2', P_3', G_1', G_2', G_3', y C_1' en la forma de AND-OR invertida y dibuje el circuito de arrastre posterior de dos niveles para este circuito integrado. [*Sugerencia:* use el método de ecuación-sustitución de la forma como se hizo en el texto al deducir la Figura 5-4, pero usando la función AND-OR invertida dada en (a) por C_{i+1}.]

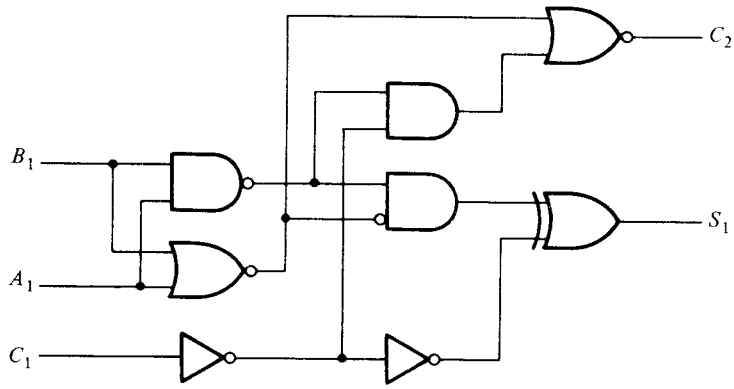


Figura P5-6 Primera etapa de un sumador paralelo

5-7. (a) Asuma que la compuerta OR-exclusiva tiene una demora de propagación de 20 ns y que las compuertas AND y OR tienen una demora de propagación de 10 ns. ¿Cuál es el tiempo total de demora de propagación en el sumador de 4 bits de la Figura 5-5?

(b) Asuma que C_5 se propaga en el recuadro de la Figura 5-5 al mismo tiempo que otros bits de arrastre (ver Problema 5-4). ¿Cuál será el tiempo de demora de propagación del sumador de 16 bits del Problema 5-2?

5-8. Diseñe un multiplicador binario que multiplique un número de 4 bits $B = b_3 b_2 b_1 b_0$ por un número de 3 bits $A = a_2 a_1 a_0$ para formar el producto $C = c_6 c_5 c_4 c_3 c_2 c_1 c_0$. Esto puede lograrse con 12 compuertas y dos sumadores paralelos de 4 bits. Las compuertas AND se usan para formar los productos en pares de bits. Por ejemplo, el producto de a_0 y b_0 pueden generarse sacando la función AND de a_0 con b_0. Los productos parciates formados por las compuertas AND se suman con los sumadores paralelos.

5-9. ¿Cuántas entradas de no importa hay en un sumador BDC?

5-10. Diseñe un circuito combinacional que genere el complemento de 9 del dígito BDC.

5-11. Diseñe una unidad aritmética decimal con dos variables de selección, V_1 y V_0 y dos dígitos BDC, A y B. La unidad debe tener cuatro operaciones aritméticas que dependen de los valores de las variables de selección de la manera como se muestra a continuación.

V_1	V_0	Función de salida
0	0	A + 9's complemento de B
0	1	$A + B$
1	0	A + 10's complemento de B
1	1	$A + 1$ (agregue 1 a A)

Use funciones MSI en el diseño y el complementador de 9 del Problema 5-10.

5-12. Es necesario diseñar un sumador decimal de dos dígitos representados en un código de exceso 3 (Tabla 1-2). Demuestre que la corrección después de sumar los dos dígitos con un sumador binario de 4 bits es de la siguiente manera:

(a) El arrastre de salida es igual al bit de arrastre del sumador binario.

(b) Si el arrastre de salida = 1, agregar 0011.

(c) Si el arrastre de salida = 0, agregar 1101.

Construya el sumador con dos sumadores binarios de 4 bits y un inversor.

5-13. Diseñe un circuito que compare dos números de 4 bits A y B, para constatar si ellos son iguales. El circuito tiene una salida x, tal que $x = 1$ si $A = B$ y $x = 0$ si $A \neq B$.

5-14. El circuito integrado 74L85 es un comparador de magnitud de 4 bits similar al de la Figura 5-7, excepto que tiene tres entradas más y circuitos internos que configuran el equivalente lógico mostrado en la Figura P5-14. Por medio de estos circuitos integrados, se pueden comparar los números de mayor longitud al conectar los comparadores en cascada. Las salidas $A < B$, $A > B$ y $A = B$ de una etapa que contenga bits menos significativos que se conectan a las correspondientes entradas $A < B$, $A > B$ y $A = B$ de la siguiente etapa que manipula bits más significativos. La etapa que manipula los bits menos significativos debe ser como el circuito mostrado en la Figura 5-7. Si se usa el CI 74L85, se debe aplicar un 1 a la entrada $A = B$ y un 0 a las entradas $A < B$ y $A > B$ en el CI que manipula los cuatro bits menos significativos. Usando un circuito como el de la Figura 5-7 y un CI 74L85, obtenga un circuito para comparar dos números de 8-bits. Justifique la operación del circuito.

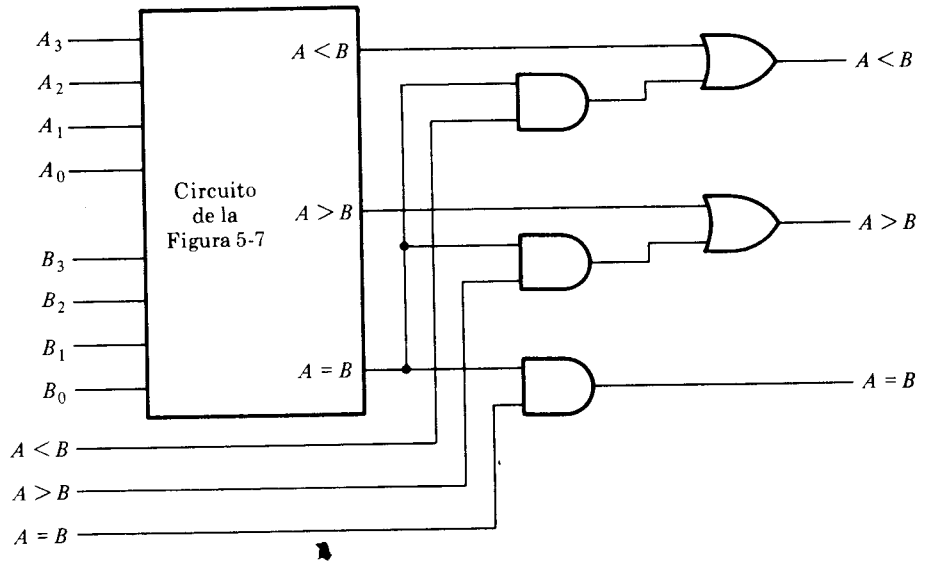


Figura P5-14 Circuito equivalente lógicamente al CI tipo 74L85

5-15. Modifique el decodificador de BDC a decimal de la Figura 5-10 para obtener una salida de sólo ceros cuando ocurra una combinación de entrada inválida.

5-16. Diseñe un convertidor de código BDC a exceso 3 con un decodificador BDC a decimal y cuatro compuertas OR.

5-17. Un circuito combinacional se define por medio de las tres siguientes funciones:

$$F_1 = x'y' + xyz'$$
$$F_2 = x' + y$$
$$F_3 = xy + x'y'$$

Diseñe un circuito con un decodificador y compuertas externas.

5-18. Un circuito combinacional se define por medio de las dos siguientes funciones:

$$F_1(x, y) = \Sigma(0, 3)$$
$$F_2(x, y) = \Sigma(1, 2, 3)$$

Configure el circuito combinacional por medio del decodificador mostrado en la Figura 5-12 y compuertas NO-Y externas.

5-19. Construya un decodificador de 5×32 con cuatro decodificadores demultiplexores de 3×8 y un decodificador de 2×4. Use la construcción de diagrama de bloque de la Figura 5-14.

5-20. Dibuje el diagrama lógico de un decodificador demultiplexor de 2 a 4 líneas usando solamente compuertas NO-O.

5-21. Especifique la tabla de verdad de un decodificador de prioridad de octal a binario. Coloque una salida para indicar que al menos una de las entradas es 1. La tabla puede ser listada con 9 filas y algunas de las entradas pueden tener valores de no importa.

5-22. Diseñe un codificador de prioridad de 4 a 2 líneas. Incluya una salida E para indicar que al menos una de las entradas es 1.

5-23. Configure la función de Boole del Ejemplo 5-4 con un multiplexor de 8×1 con A, B y D conectados para seleccionar las líneas s_2, s_1 y s_0 respectivamente.

5-24. Configure el circuito combinacional especificado en el Problema 5-17 con un doble multiplexor de 4 a 1 línea, una compuerta O y un inversor.

5-25. Obtenga un multiplexor de 8×1 con un doble multiplexor de 4 a 1 línea con entradas de habilitación (enable) separados pero con líneas de selección comunes. Use la construcción por diagrama de bloque.

5-26. Configure un circuito sumador completo con multiplexores.

5-27. La ROM de 32×6 conjuntamente con la línea 2^0 como se muestra en la Figura P5-27 convierte un número binario de 6 bits a su correspondiente número BDC de 2 dígitos. Por ejemplo, el binario 100001 se convierte al BDC 011 0011 (decimal 33). Especifique la tabla de verdad para la ROM.

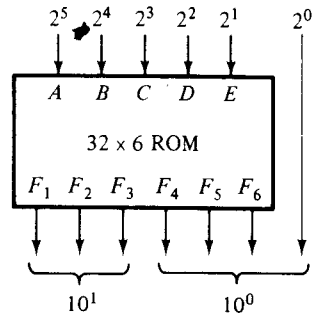


Figura P5-27 Conversor de binario a decimal

5-28. Pruebe que una ROM de 32×8 puede usarse para configurar un circuito que genere el cuadrado binario de un número de 5 bits de entrada con $B_0 = A_0$ y $B_1 = 0$. Como en la Figura 5-24(a). Dibuje el diagrama de bloque del circuito y liste las primeras y últimas entradas de la tabla de verdad de la ROM.

5-29. ¿Qué tamaño de ROM se usaría para configurar:

(a) Un sumador sustractor BDC con una entrada de control para seleccionar entre la suma y la resta?

(b) Un multiplicador binario que multiplica dos números de 4 bits?

(c) Unos multiplexores dobles de 4 a 1 línea con entradas de selección comunes?

5-30. Cada inversor de salida en el PLA de la Figura 5-26 se remplaza con una compuerta OR-exclusiva. Cada compuerta OR-exclusiva tiene dos entradas. Una entrada se conecta a la salida de la compuerta OR y la otra entrada se conecta por medio de enlaces a una señal equivalente a cero o uno. Demuestre cómo seleccionar la salida verdadera/complemento en esta configuración.

5-31. Deduzca la tabla de programación del PLA para el circuito combinacional que eleva al cuadrado un número de 3 bits. Minimice el número de términos producto. (Ver la Figura 5-24 para la configuración con ROM equivalente.)

5-32. Liste la tabla de programación del PLA para el convertidor de código de BDC a exceso 3 definido en la Sección 4-5.

Lógica secuencial

6

6-1 INTRODUCCION

Los circuitos digitales hasta ahora considerados han sido combinacionales, es decir, las salidas en un instante dado de tiempo son enteramente dependientes de las entradas presentes en ese mismo tiempo. Aunque cada sistema digital debe tener circuitos combinacionales, la mayoría de los sistemas encontrados en la práctica incluyen también elementos de memoria, los cuales requieren que el sistema se describa en términos de la *lógica secuencial*.

Un diagrama de bloque de un circuito secuencial se muestra en la Figura 6-1. Este consiste en un circuito combinacional al cual se le conectan elementos de memoria para formar un camino de realimentación. Los elementos de memoria son capaces de almacenar información binaria dentro de ellos. La información binaria almacenada en los elementos de memoria en un tiempo dado define el *estado* del circuito secuencial. El circuito secuencial recibe la información binaria de las entradas externas. Estas entradas, conjuntamente con el presente estado de los elementos de memoria, determinan el valor binario de los terminales de salida. También determinan la condición de cambio de estado en los elementos de memoria. El diagrama de bloque demuestra que las salidas externas en un circuito secuencial son una función no solamente de las entradas externas sino del presente estado de los elementos de memoria. El siguiente estado de los elementos de memoria es también una función de las entradas externas y del estado presente. Así, un circuito secuencial se especifica por medio de una secuencia de tiempo de las entradas, salidas y estados internos.

Hay dos tipos de circuitos secuenciales. Su clasificación depende del tiempo de sus señales. Un circuito secuencial *sincrónico* es un sistema cuyo comportamiento puede definirse a partir del conocimiento de sus señales en instantes discretos de tiempo. El comportamiento de un circuito *asincrónico* depende del orden en que cambien las señales de entrada y puedan ser afectadas en un instante dado de tiempo. Los elementos de memoria comúnmente usados en los circuitos secuenciales asincrónicos son mecanismos retardadores de tiempo. La capacidad de memoria de los mecanismos retardadores de tiempo se debe al hecho de que la señal

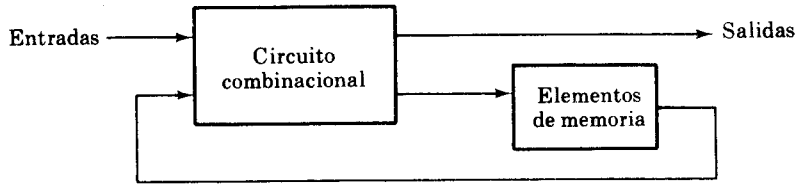


Figura 6-1 Diagrama de bloque de un circuito secuencial

gasta un tiempo finito para propagarse a través del dispositivo. En la práctica, el retardo de propagación interna de las compuertas lógicas es de una duración suficiente como para producir el retardo necesario, de tal manera que las unidades físicas de retardo de tiempo puedan ser despreciables. En los sistemas asincrónicos tipo compuerta, los elementos de memoria de la Figura 6-1 consisten en compuertas lógicas cuyos retardos de propagación constituyen la memoria requerida. Así, un circuito secuencial asincrónico puede tomarse como un circuito combinacional con realimentación. Debido a la realimentación entre las compuertas lógicas, un circuito secuencial asincrónico puede a veces volverse inestable. El problema de inestabilidad impone muchas dificultades al diseñador. Por tanto, su uso no es tan común como en los sistemas sincrónicos.

Un sistema lógico secuencial sincrónico, por definición, puede usar señales que afecten los elementos de memoria solamente en instantes de tiempo discreto. Una forma de lograr este propósito es usar pulsos de duración limitada a través del sistema de tal manera que la amplitud de un pulso represente lógica 1 y otra amplitud de pulso (o la ausencia de un pulso) represente lógica 0. La dificultad con un sistema de pulsos es que cualquier par de pulsos que lleguen de fuentes separadas independientes a las entradas de la misma compuerta mostrarán retardos no predecibles de tal manera que se separarán los pulsos ligeramente, resultando una operación no confiable.

Los sistemas lógicos secuenciales sincrónicos prácticos usan amplitudes fijas tales como niveles de voltaje para las señales binarias. La sincronización se logra por un dispositivo de tiempo llamado *generador maestro de tiempo* el cual genera un tren periódico de *pulsos de reloj*. Los pulsos de reloj se distribuyen a través del sistema de tal manera que los elementos de memoria son afectadas solamente con la llegada del pulso de sincronización. En la práctica, el pulso de reloj se aplica a las compuertas AND conjuntamente con las señales que especifican los cambios requeridos en los elementos de memoria. Las salidas de la compuerta AND pueden trasmitir señales solamente en los instantes que coinciden con la llegada de los pulsos de reloj. Los circuitos secuenciales sincrónicos que usan pulsos de reloj en las entradas de los elementos de memoria se llaman *circuitos secuenciales temporizados*. Los circuitos secuenciales temporizados son el tipo más comúnmente usado. No presentan problemas de inestabilidad y su temporización se divide fácilmente en pasos discretos independientes, cada uno de los cuales se considera separadamente. Los circuitos secuenciales que se discuten en este libro son exclusivamente del tipo temporizado.

Los elementos de memoria usados en los circuitos secuenciales temporizados se llaman *flip-flops*. Estos circuitos son celdas binarias capaces de almacenar un bit de información. Un circuito flip-flop tiene dos entradas, una para el valor normal y uno para el valor complemento del bit almacenado en él. La información binaria puede entrar a un flip-flop en una variedad de formas, hecho éste, que determina diferentes tipos de flip-flops. En la siguiente sección se examinan varios tipos de flip-flops y se definen sus propiedades lógicas.

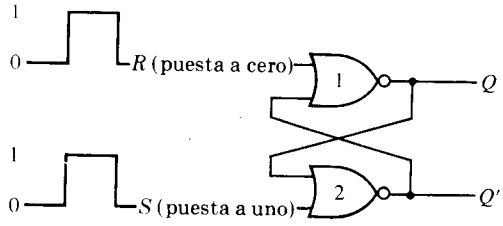
6-2 FLIP-FLOPS

Un circuito flip-flop puede mantener un estado binario indefinidamente (siempre y cuando se esté suministrando potencia al circuito) hasta que se cambie por una señal de entrada para cambiar estados. La principal diferencia entre varios tipos de flip-flops es el número de entradas que poseen y la manera en la cual las entradas afectan el estado binario. Los tipos de flip-flops más comunes se discuten a continuación.

Circuito básico de un flip-flop

Se mencionó en las Secciones 4-7 y 4-8 que un circuito flip-flop puede construirse con dos compuertas NAND o dos compuertas NOR. Estas construcciones se muestran en los diagramas lógicos de las Figuras 6-2 y 6-3. Cada circuito forma un flip-flop básico del cual se puede construir uno más complicado. La conexión de acoplamiento intercruzado de la salida de una compuerta a la entrada de la otra constituye un camino de realimentación. Por esta razón, los circuitos se clasifican como circuitos secuenciales asincrónicos. Cada flip-flop tiene dos salidas, Q y Q' y dos entradas S (*set*) y R (*reset*). Este tipo de flip-flop se llama *flip-flop RS acoplado directamente* o *bloqueador SR* (SR latch). Las letras R y S son las iniciales de los nombres en inglés de las entradas (reset, set).

Para analizar la operación del circuito de la Figura 6-2 se debe recordar que la salida de una compuerta NOR es 0 si cualquier entrada es 1 y que la salida es 1 solamente cuando todas las entradas sean 0. Como punto de partida asúmase que la entrada de puesta a uno (set) es 1 y que la entrada de puesta a cero (reset) sea 0. Como la compuerta 2 tiene una entrada de 1, su salida Q' debe ser 0, lo cual coloca ambas entradas



(a) Diagrama lógico

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(después de $S = 1$, $R = 0$)
0	1	0	1	
0	0	0	1	(después de $S = 0$, $R = 1$)
1	1	0	0	

(b) Tabla de verdad

Figura 6-2 Circuito flip-flop básico con compuertas NOR

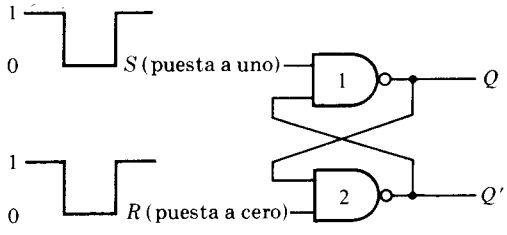
de la compuerta 1 a 0 para tener la salida Q como 1. Cuando la entrada de puesta a uno (set) vuelva a 0, las salidas permanecerán iguales ya que la salida Q permanece como 1, dejando una entrada de la compuerta 2 en 1. Esto causa que la salida Q' permanezca en 0 lo cual coloca ambas entradas de la compuerta número 1 en 0 y así la salida Q es 1. De la misma manera es posible demostrar que un 1 en la entrada de puesta a cero (reset) cambia la salida Q a 0 y Q' a 1. Cuando la entrada de puesta a cero cambia a 0, las salidas no cambian.

Cuando se aplica un 1 a ambas entradas de puesta a uno y puesta a cero ambas salidas Q y Q' van a 0. Esta condición viola el hecho de que las salidas Q y Q' son complementos entre sí. En operación normal esta condición debe evitarse asegurándose que no se aplica un 1 a ambas entradas simultáneamente.

Un flip-flop tiene dos entradas útiles. Cuando $Q = 1$ y $Q' = 0$ estará en el estado de *puesta a uno* (o estado 1). Cuando $Q = 0$ y $Q' = 1$ estará en el estado de puesta a cero (o estado 0). Las salidas Q y Q' son complementos entre sí y se les trata como salidas normales y de complemento respectivamente. El estado binario de un flip-flop se toma como el valor de su salida normal.

Bajo operación normal, ambas entradas permanecen en 0 a no ser que el estado del flip-flop haya cambiado. La aplicación de un 1 momentáneo a la entrada de puesta a uno causará que el flip-flop vaya a ese estado. La entrada de puesta a uno debe volver a cero antes que se aplique un 1 a la entrada de puesta a cero. Un 1 momentáneo aplicado a la entrada de puesta a cero causará que el flip-flop vaya al *estado de borrado* (o puesta a cero). Cuando ambas entradas son inicialmente cero y se aplica un 1 a la entrada de puesta a uno mientras que el flip-flop esté en el estado de puesta a uno o se aplica un 1 a la entrada de puesta a cero mientras que el flip-flop esté en el estado de borrado, quedarán las salidas sin cambio. Cuando se aplica un 1 a ambas entradas de puesta a uno y de puesta a cero, ambas salidas irán a 0. Este estado es indefinido y se evita normalmente. Si ahora ambas salidas van a 0, el estado del flip-flop es indeterminado y depende de aquella entrada que permanezca por mayor tiempo en 1 antes de hacer la transición a 0.

El circuito flip-flop básico NAND de la Figura 6-3 opera con ambas entradas normalmente en 1 a no ser que el estado del flip-flop tenga que cambiarse. La aplicación de un 0 momentáneo a la entrada de puesta a



(a) Diagrama lógico

S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(después de $S = 1$, $R = 0$)
0	1	1	0	
1	1	1	0	(después de $S = 0$, $R = 1$)
0	0	1	1	

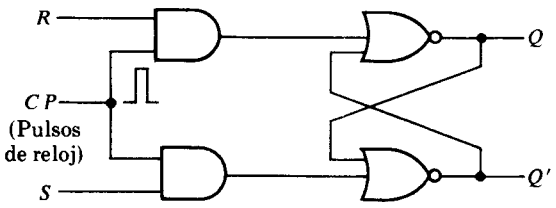
(b) Tabla de verdad

Figura 6-3 Circuito flip-flop básico con compuertas NAND

uno, causará que Q vaya a 1 y Q' vaya a 0, llevando el flip-flop al estado de puesta a uno. Después que la entrada de puesta a uno vuelva a 1, un 0 momentáneo en la entrada de puesta a cero causará la transición al estado de borrado (clear). Cuando ambas entradas vayan a 0, ambas salidas irán a 1; esta condición se evita en la operación normal de un flip-flop.

Flip-flop *RS* temporizado

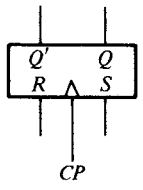
El flip-flop básico por sí solo es un circuito secuencial asincrónico. Agregando compuertas a las entradas del circuito básico, puede hacerse que el flip-flop responda a los niveles de entrada durante la ocurrencia del pulso del reloj. El flip-flop *RS* temporizado mostrado en la Figura 6-4(a) consiste en un flip-flop básico NOR y dos compuertas AND. Las salidas de dos compuertas AND permanecen en cero mientras el pulso del reloj (abreviado en inglés *CP*) sea 0, independientemente de los valores de entrada de S y R. Cuando el pulso del reloj vaya a 1, la información de las entradas S y R se permite llegar al flip-flop básico. El estado de puesta a uno se logra con $S = 1$, $R = 0$ y $CP = 1$. Para cambiar el estado de puesta a cero (o borrado) las entradas deben ser $S = 0$, $R = 1$ y $CP = 1$. Con $S = 1$ y $R = 1$, la ocurrencia de los pulsos de reloj causará que ambas salidas vayan momentáneamente a 0. Cuando se quite el pulso, el estado del flip-flop será indeterminado, es decir, podría resultar cualquier estado,



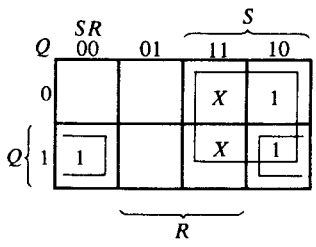
(a) Diagrama lógico

Q	S	R	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminado
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminado

(c) Tabla característica



(b) Símbolo gráfico



$Q(t+1) = S + R'Q$

$SR = 0$

(d) Ecuación característica

Figura 6-4 Flip-flop *RS* temporizado

dependiendo de si la entrada de puesta a uno o la de puesta a cero del flip-flop básico, permanezca el mayor tiempo, antes de la transición a 0 al final del pulso.

El símbolo gráfico del flip-flop *RS* sincronizado se muestra en la Figura 6-4(b). Tiene tres entradas: *S*, *R* y *CP*. La entrada *CP* no se escribe dentro del recuadro debido a que se reconoce fácilmente por un pequeño triángulo. El triángulo es un símbolo para el *indicador dinámico* y denota el hecho de que el flip-flop responde a una *transición* del reloj de entrada o flanco de subida de una señal de un nivel bajo (o binario) a un nivel alto (1 binario). Las salidas del flip-flop se marcan con *Q* y *Q' dentro* del recuadro. Se le puede asignar al flip-flop un nombre de variable diferente aunque se escriba una *Q* dentro del recuadro. En este caso la letra escogida para la variable del flip-flop se marca *por fuera* del recuadro y a lo largo de la línea de salida. El estado del flip-flop se determina del valor de su salida normal *Q*. Si se desea obtener el complemento de la salida normal, no es necesario usar un inversor ya que el valor complementado se obtiene directamente de la salida *Q'*.

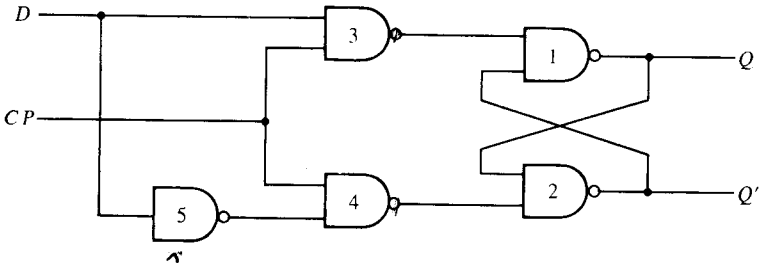
La tabla característica del flip-flop se muestra en la Figura 6-4(c). Esta tabla resume la operación del flip-flop en forma de tabulado. *Q* es el estado binario del flip-flop en un tiempo dado (refiriéndose al *estado presente*), las columnas *S* y *R* dan los valores posibles de las entradas y $Q(t+1)$ es el estado del flip-flop después de la ocurrencia de un pulso de reloj (refiriéndose al *siguiente estado*).

La ecuación característica de un flip-flop se deduce del mapa de la Figura 6-4(d). Esta ecuación especifica el valor del siguiente estado como una función del presente estado y de las entradas. La ecuación característica es una expresión algebraica para la información binaria de la tabla característica. Los dos estados indeterminados se marcan con una X en el mapa, ya que pueden resultar como 1 o como 0. Sin embargo la relación $SR = 0$ debe incluirse como parte de la ecuación característica para especificar que *S* y *R* no pueden ser iguales a 1 simultáneamente.

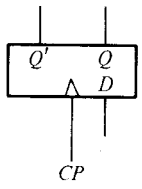
Flip-flop *D*

El flip-flop *D* mostrado en la Figura 6-5 es una modificación del flip-flop *RS* sincronizado. Las compuertas NAND 1 y 2 forman el flip-flop básico y las compuertas 3 y 4 las modifican para conformar el flip-flop *RS* sincronizado. La entrada *D* va directamente a la entrada *S* y su complemento se aplica a la entrada *R* a través de la compuerta 5. Mientras que el pulso de reloj de entrada sea un 0, las compuertas 3 y 4 tienen un 1 en sus salidas, independientemente del valor de las otras entradas. Esto está de acuerdo a los requisitos de que las dos entradas del flip-flop básico NAND (Figura 6-3) permanezcan inicialmente en el nivel de 1. La entrada *D* se comprueba durante la ocurrencia del pulso de reloj. Si es 1, la salida de la compuerta 3 va a 0, cambiando el flip-flop al estado de puesta a uno (a no ser que ya esté en ese estado). Si es 0, la salida de la compuerta 4 va a 0, cambiando el flip-flop al estado de borrado.

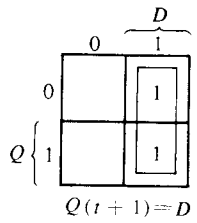
El flip-flop tipo *D* recibe su nombre por la habilidad de trasmitir "datos" a un flip-flop. Es básicamente un flip-flop *RS* con un inversor en



(a) Diagrama lógico con compuertas NAND



Q	D	$Q(t+1)$
0	0	0
0	1	1
1	0	0
1	1	1



(b) Símbolo gráfico (c) Tabla característica (d) Ecuación característica

Figura 6-5 Flip-flop D temporizado

la entrada R. El inversor agregado reduce el número de entradas de dos a uno. Este tipo de flip-flop se llama algunas veces *bloqueador D con compuertas* o *flip-flop de bloqueo*. La entrada CP se le da a menudo la designación variable G (de *gate*) para indicar que esta entrada habilita el flip-flop de bloqueo para hacer posible que los datos entren al mismo.

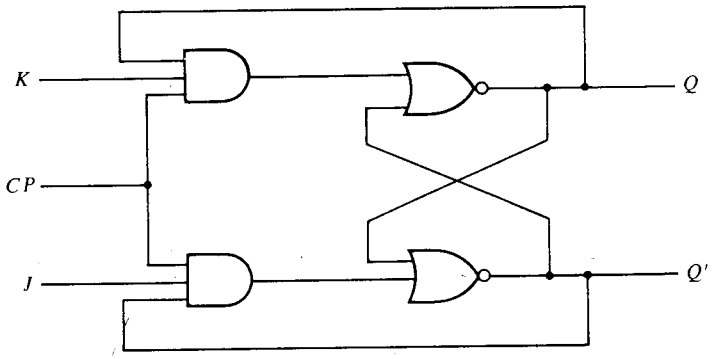
El símbolo para el flip-flop D sincronizado se muestra en la Figura 6-5(b). La tabla característica se lista en la parte (c) y la ecuación característica se deriva en la parte (d). La ecuación característica muestra que el siguiente estado del flip-flop es igual a la entrada D y es independiente del valor del presente estado.

Flip-flop *JK*

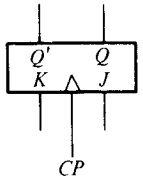
Un flip-flop JK es un refinamiento del flip-flop RS ya que el estado indeterminado del tipo RS se define en el tipo JK. Las entradas J y K se comportan como las entradas S y R para poner a uno o cero (set o clear) al flip-flop (nótese que en el flip-flop JK la letra J se usa para la entrada de *puesta a uno* y la letra K para la entrada de *puesta a cero*). Cuando ambas entradas se aplican a J y K simultáneamente, el flip-flop cambia a su estado de complemento, esto es, si $Q = 1$ cambia a $Q = 0$ y viceversa.

Un flip-flop JK sincronizado se muestra en la Figura 6-6(a). La salida Q se aplica con K y CP a una compuerta AND de tal manera que el flip-flop se ponga a cero (clear) durante un pulso de reloj solamente si Q fue 1 previamente. De manera similar la salida Q' se aplica con J y CP a una compuerta AND de tal manera que el flip-flop se ponga a uno con un pulso de reloj, solamente si Q' fue 1 previamente.

Como se muestra en la tabla característica en la Figura 6-6(c), el flip-flop JK se comporta como un flip-flop RS excepto cuando J y K sean



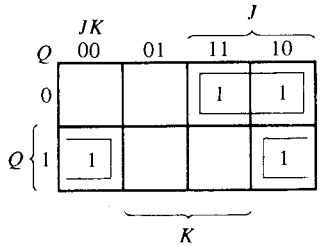
(a) Diagrama lógico



(b) Símbolo gráfico

Q	J	K	$Q(t+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(c) Tabla característica



$Q(t+1) = JQ' + K'Q$

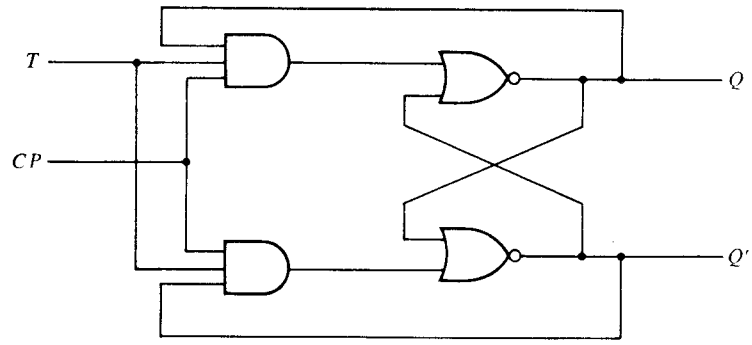
(d) Ecuación característica

Figura 6-6 Flip-flop JK temporizado

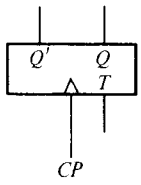
ambos 1. Cuando J y K sean 1, el pulso de reloj se trasmite a través de una compuerta AND solamente; aquella cuya entrada se conecta a la salida del flip-flop la cual es al presente igual a 1. Así, si $Q = 1$, la salida de la compuerta AND superior se convertirá en 1 una vez se aplique un pulso de reloj y el flip-flop se ponga a cero. Si $Q' = 1$ la salida de la compuerta AND se convierte en 1 y el flip-flop se pone a uno. En cualquier caso, el estado de salida del flip-flop se complementa.

Las entradas en el símbolo gráfico para el flip-flop JK deben marcarse con una J (debajo de Q) y K (debajo de Q'). La ecuación característica se da en la Figura 6-4(d) y se deduce del mapa de la tabla característica.

Nótese que debido a la conexión de realimentación del flip-flop JK, la señal CP que permanece en 1 (mientras que $J = K = 1$) causará transiciones repetidas y continuas de las salidas después de que las salidas hayan sido complementadas. Para evitar esta operación indeseable, los pulsos de reloj deben tener un tiempo de duración que es menor que la demora de propagación a través del flip-flop. Esta es una restricción, ya que la operación del circuito depende del ancho de los pulsos. Por esta razón los flip-flops JK nunca se construyen como se muestra en la Figura 6-6(a). La restricción del ancho del pulso puede ser eliminada con un maestro esclavo o una construcción activada por flanco de la manera discutida en la siguiente sección. El mismo razonamiento se aplica al flip-flop T presentado a continuación.



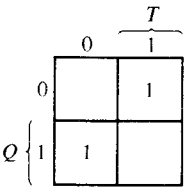
(a) Diagrama lógico



(b) Símbolo gráfico

Q	T	$Q(t+1)$
0	0	0
0	1	1
1	0	1
1	1	0

(c) Tabla característica



$Q(t+1) = TQ' + T'Q$

(d) Ecuación característica

Figura 6-7 Flip-flop T temporizado

Flip-flop T

El flip-flop T es la versión de una entrada, del flip-flop JK. Como se muestra en la Figura 6-7(a), el flip-flop J se obtiene de un tipo JK a la cual se le unen las dos entradas. El nombre T se deriva de la habilidad del flip-flop de variar ("toggle") o cambiar estado. Independientemente del presente estado del flip-flop, este asume el estado de complemento cuando ocurre el pulso de reloj mientras que la entrada T esté en lógica 1. El símbolo, la tabla característica y la ecuación característica del flip-flop T se muestran en la Figura 6-7, partes (b), (c) y (d) respectivamente.

Los flip-flops introducidos en esta sección son los de tipo más común comercialmente. Los procedimientos de análisis y de diseño desarrollados en este capítulo se aplican a cualquier flip-flop sincronizado una vez que se haya definido su tabla característica.

6-3 DISPARO DE LOS FLIP-FLOPS (TRIGGERING)

El estado de un flip-flop se varía debido a un cambio momentáneo en la señal de entrada. Este cambio momentáneo se le llama *disparo* (trigger) y la transición que lo causa se dice que dispara el flip-flop. Los flip-flops asincrónicos, tales como los circuitos básicos de la Figura 6-2 y 6-3, requieren un disparo de entrada definido por un cambio de *nivel* de señal.

Este nivel debe regresarse a un valor inicial (0 en el flip-flop a base de NOR y 1 en aquella a base de NAND) antes de aplicarle el segundo disparo. Los flip-flops sincronizados se disparan por medio de *pulsos*. Un pulso comienza a partir de su valor inicial de 0, va momentáneamente a 1 y después de un corto período, regresa a su valor inicial 0. El intervalo de tiempo que ocurre desde la aplicación del pulso hasta que ocurra la transición de salida, es un factor crítico que requiere investigación posterior.

Como se ve en el diagrama de bloque de la Figura 6-1, un circuito secuencial tiene un camino de realimentación entre el circuito combinacional y los elementos de memoria. Este camino puede producir inestabilidad si las salidas de los elementos de memoria (flip-flops) están cambiando mientras que las salidas del circuito combinacional que van a las entradas de los flip-flops estén siendo sometidas a disparo por el pulso del reloj. El problema de tiempo puede ser prevenido si las salidas de los flip-flops no comienzan a cambiar hasta que el impulso de entrada haya retornado a 0. Para asegurar tal operación, un flip-flop debe tener un retardo de propagación de la señal desde la entrada hasta la salida, en exceso, con respecto a la duración del pulso. Este retardo es comúnmente muy difícil de controlar si el diseñador depende totalmente del retardo de propagación de las compuertas lógicas. Una forma de asegurar el retardo adecuado es incluir dentro del circuito del flip-flop una unidad de retardo físico que tenga un retardo igual o mayor que la duración del pulso. Una forma muy buena de resolver el problema de temporización por realimentación es hacer el flip-flop sensible a la *transición* del pulso en vez de la duración del pulso.

Un pulso de reloj puede ser positivo o negativo. Una fuente de reloj positiva permanece en 0 durante el intervalo entre los pulsos y va a 1 durante la ocurrencia de un pulso. El pulso pasa por dos transiciones de señal: de 0 a 1 y el regreso de 1 a 0. Como se ve en la Figura 6-8, la transición positiva se define como *flanco positivo* y la transición negativa como *flanco negativo*. Esta definición se aplica a los pulsos negativos.

Los flip-flops sincronizados que se introdujeron en la Sección 6-2 se disparan durante el flanco positivo del pulso y el estado de transición comienza tan pronto como el pulso alcanza el nivel de lógica 1. El nuevo estado del flip-flop puede aparecer en los terminales de salida mientras

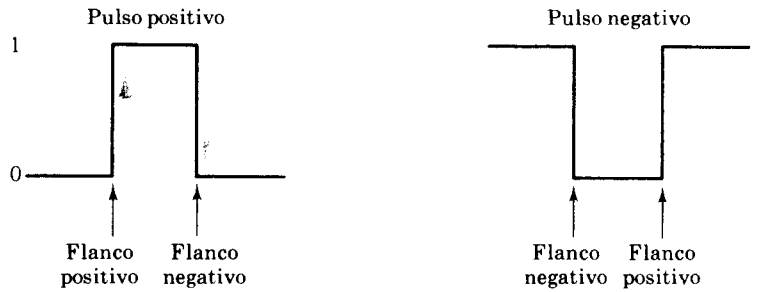


Figura 6-8 Definición de la transición de un pulso de reloj

que el pulso de entrada sea 1 todavía. Si las otras entradas del flip-flop cambian mientras que el pulso sea 1, el flip-flop empezará a responder a esos valores nuevos y puede ocurrir un nuevo estado de salida. Cuando esto pasa, la salida de un flip-flop no puede ser aplicada a las entradas de otro flip-flop cuando ambos sean disparados por el mismo pulso de reloj. Sin embargo, si se puede hacer que el flip-flop responda al flanco positivo (o negativo) de transición *solamente*, en vez de la duración total del pulso, entonces se puede eliminar el problema de la múltiple transición.

Una manera de hacer que el flip-flop responda solamente al pulso de transición es usar un acoplamiento capacitivo. En esta configuración, se inserta un circuito RC (resistencia-condensador) en la entrada de reloj del flip-flop. Este circuito genera un pico en respuesta al cambio momentáneo de la señal de entrada. Un flanco positivo emerge de tal circuito con un pico positivo y un flanco negativo con un pico negativo (spike). La activación de los flancos se logra diseñando el flip-flop para ignorar un pico y dispararse con la ocurrencia del siguiente. Otra forma de lograr el disparo de los flancos es el uso de un maestro esclavo o flip-flop de disparo por flancos como se discute a continuación.

Flip-flop maestro esclavo

Un flip-flop maestro esclavo se construye con dos flip-flops separados. Un circuito sirve como maestro y el otro como esclavo y el circuito completo se trata como un *flip-flop maestro esclavo*. El diagrama lógico de un flip-flop maestro esclavo RS se muestra en la Figura 6-9. Esta consiste en un flip-flop maestro, un esclavo y un inversor. Cuando el pulso de reloj CP es 0, la salida del inversor es 1. Como el pulso de entrada de reloj del esclavo es 1, el flip-flop se habilita y la salida Q es igual a Y mientras que Q' se iguala a Y'. El flip-flop maestro se inhabilita debido a que $CP = 0$. Cuando el pulso de reloj se convierte en 1, la información en las entradas externas R y S se trasmiten al flip-flop maestro. El flip-flop maestro sin embargo, se aísla por el intervalo en que el pulso esté en un nivel de 1, ya que la salida del inversor es 0. Cuando el pulso regresa a 0, el flip-flop

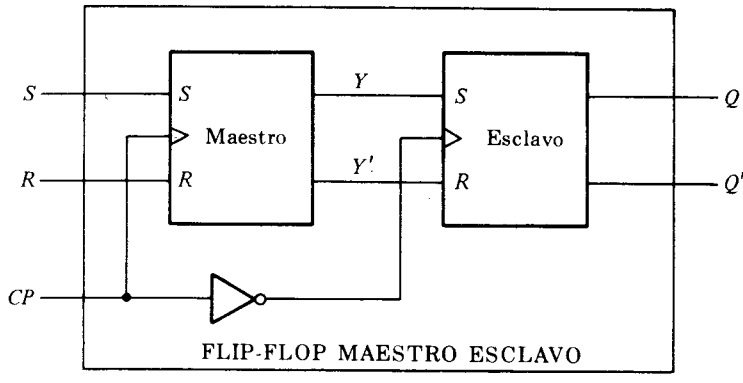


Figura 6-9 Diagrama lógico de un flip-flop maestro esclavo

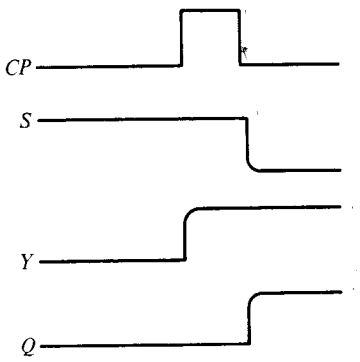


Figura 6-10 Relaciones de tiempo de un flip-flop maestro esclavo

maestro se aísla, lo cual previene que las entradas externas lo afecten. El flip-flop esclavo irá al mismo estado que el maestro.

Las relaciones de tiempo mostradas en la Figura 6-10 ilustran la secuencia de eventos que ocurren en un flip-flop maestro esclavo. Asúmase que el flip-flop está en el estado de puesta a cero antes de la ocurrencia de un pulso, de tal manera que $Y = 0$ y $Q = 0$. Las condiciones de entrada son $S = 1$, $R = 0$ y el siguiente pulso de reloj debe conmutar el flip-flop al estado de puesta a uno con $Q = 1$. Durante la transación del pulso de 0 a 1, el flip-flop maestro se pone a uno y conmuta Y a 1. El flip-flop esclavo no se afecta debido a que su CP es 0. Como el flip-flop maestro es un circuito interno, su cambio de estado no se nota en las salidas Q y Q'. Cuando el pulso regrese a 0, la información del maestro se permite pasar al esclavo haciendo la salida externa $Q = 1$. Nótese que la entrada externa S puede cambiarse al mismo tiempo que el pulso va a través de la transición de un flanco negativo. Esto se debe a que una vez que CP alcance el 0, el maestro se inhabilita y sus entradas R y S no tienen influencia hasta que el siguiente pulso de reloj ocurra. Entonces, en un flip-flop maestro esclavo, es posible variar la salida y la información de entrada, con el mismo pulso de reloj. Se debe tener en cuenta que la entrada S podría venir de la salida de otro flip-flop maestro esclavo que fuera conmutado con el mismo pulso de reloj.

El comportamiento del flip-flop maestro esclavo ya descrito determina que los cambios de estado en todos los flip-flops coincide con la transición del flanco negativo del pulso. Sin embargo, algunos flip-flops maestro esclavo de CI cambian los estados de salida en la transición del flanco positivo de los pulsos de reloj. Esto ocurre en los flip-flops que tienen un inversor adicional entre el terminal CP y la entrada del maestro. Este tipo de flip-flops son disparados con pulsos negativos (ver Figura 6-8), tales que el flanco negativo del pulso afecta al maestro y el flanco positivo afecta al esclavo y a los terminales de salida.

La combinación maestro esclavo puede contruirse para cualquier tipo de flip-flops agregando un flip-flop RS sincronizado con un reloj invertido para formar un esclavo. Un ejemplo de un flip-flop JK maestro esclavo construido con compuertas NAND se muestra en la Figura 6-11. Este consiste en dos flip-flops; las compuertas 1 hasta 4 forman el flip-flop

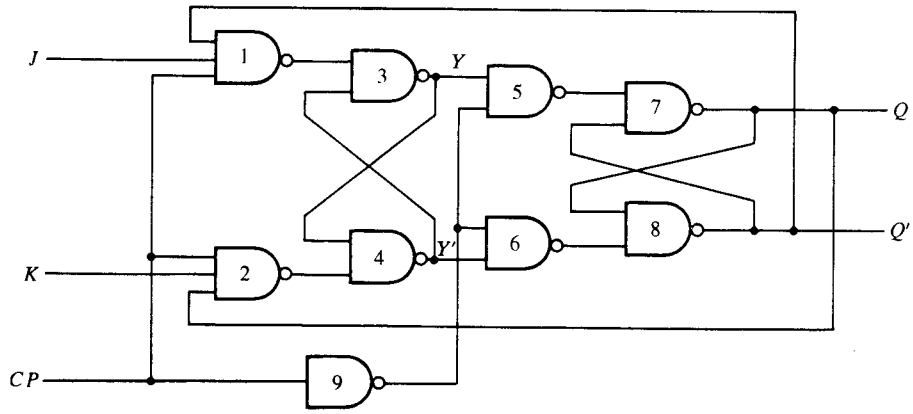


Figura 6-11 Flip-flop *JK* temporizado maestro esclavo

maestro y las compuertas 5 hasta 8 forman el flip-flop esclavo. La información presente en las entradas J y K se trasmite al flip-flop maestro en el flanco positivo del pulso de reloj y se sostiene allí hasta que el flanco negativo del pulso de reloj sucede, después del cual se permite pasar hasta el flip-flop esclavo. El reloj de entrada es normalmente 0, lo cual mantiene las salidas de las compuertas 1 y 2 en el nivel de 1. Esto previene a las entradas J y K de afectar el flip-flop maestro. El flip-flop esclavo es del tipo RS temporizado con el flip-flop maestro que suministra las entradas y el reloj de entrada invertido por la compuerta 9. Cuando el reloj es 0, la salida de la compuerta 9 es 1 de manera que la salida Q es igual a Y y Q' es igual a Y'. Cuando ocurre el flanco positivo de un pulso de reloj, el flip-flop maestro se afecta y puede conmutar estados. El flip-flop esclavo se aísla durante el tiempo en que el reloj esté en el nivel 1, debido a que la salida de la compuerta 9 suministra un 1 a ambas entradas del flip-flop básico NAND de las compuertas 7 y 8. Cuando el reloj de entrada regrese a 0, el flip-flop maestro se aísla de las entradas J y K y el flip-flop esclavo va al mismo estado del flip-flop maestro.

Considérese un sistema digital que contenga muchos flip-flops maestro esclavo, con las salidas de algunos flip-flops conectados a las entradas de otros. Asúmase que las entradas del pulso de reloj a todos los flip-flops están sincronizados (ocurren al mismo tiempo). Al comienzo de cada pulso de reloj, algunos de los elementos maestro cambian de estado, pero todos los flip-flops de salida permanecen en sus valores previos. Después que el pulso de reloj regrese a 0, algunas de las salidas cambian de estado, pero ninguno de estos estados nuevos tienen un efecto en cualquiera de los elementos maestro hasta el siguiente pulso de reloj. Así, los estados de los flip-flops en el sistema pueden cambiarse simultáneamente durante el mismo pulso de reloj, aunque las salidas de los flip-flops se conectan a las entradas de otros. Esto es posible porque el nuevo estado aparece en los terminales de salida solamente después que el pulso de reloj haya cambiado a cero. Por tanto el contenido binario de un flip-flop puede trasferirse al segundo y el contenido del segundo trasferirse al primero y ambas trasferencias ocurren durante el mismo pulso de reloj.

Flip-flop disparado por flanco

Otro tipo de flip-flop que sincroniza los cambios de estado durante una transición de pulso de reloj es el flip-flop *disparado por flanco* (edge-triggered flip-flop). En este tipo de flip-flop, las transiciones de salida ocurren en un nivel específico del pulso de reloj. Cuando el nivel de entrada del pulso excede este umbral, se cierran las entradas y el flip-flop es por tanto inactivo a cambios posteriores en las entradas hasta que el pulso de reloj regrese a cero y ocurra otro pulso. Algunos flip-flops disparados por flanco causan una transición en el flanco positivo del pulso y otras causan una transición en el flanco negativo del pulso.

El diagrama lógico de un flip-flop tipo D disparado por flanco positivo se muestra en la Figura 6-12. Este consiste en tres flip-flops básicos del tipo mostrado en la Figura 6-3. Las compuertas NAND 1 y 2 constituyen un flip-flop básico y las compuertas 3 y 4 otro. El tercer flip-flop básico que comprende las compuertas 5 y 6 suministra las salidas del circuito. Las entradas S y R del tercer flip-flop básico deben mantenerse en lógica 1 para que las salidas permanezcan en sus valores estables. Cuando $S = 0$ y $R = 1$, la salida va al estado de puesta a uno con $Q = 1$. Cuando $S = 1$ y $R = 0$, la salida va al estado de puesta a cero con $Q = 0$. Las salidas S y R se determinan de los estados de los otros dos flip-flops básicos. Estos dos flip-flops básicos responden a las entradas externas D (datos) y a CP (pulso de reloj).

La operación del circuito se explica en la Figura 6-13 donde las compuertas 1-4 se redibujan para mostrar todas las transiciones posibles. Las salidas S y R de las compuertas 2 y 3 van a las compuertas 5 y 6 como se muestra en la Figura 6-12, para suministrar las salidas actuales del flip-flop. La Figura 6-13(a) muestra los valores binarios de las salidas de las cuatro compuertas cuando $CP = 0$. La entrada D bien podría ser igual

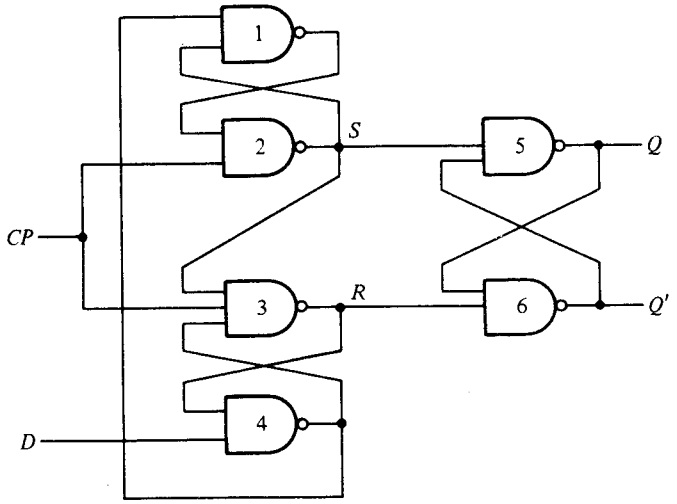
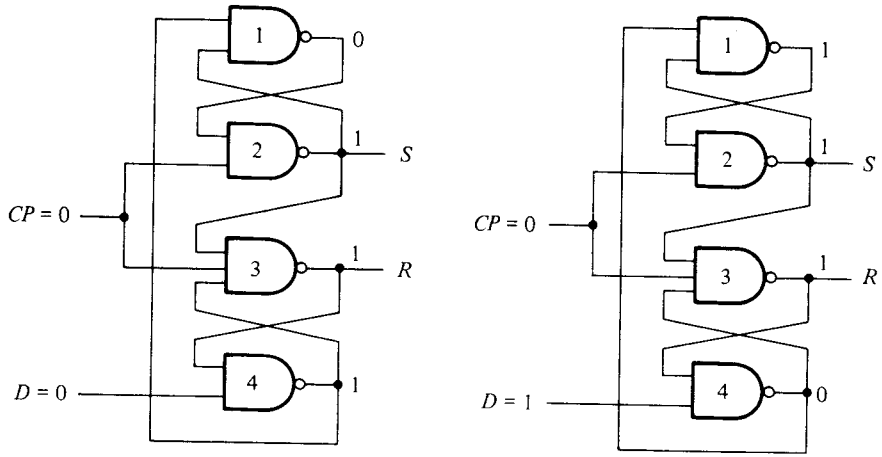
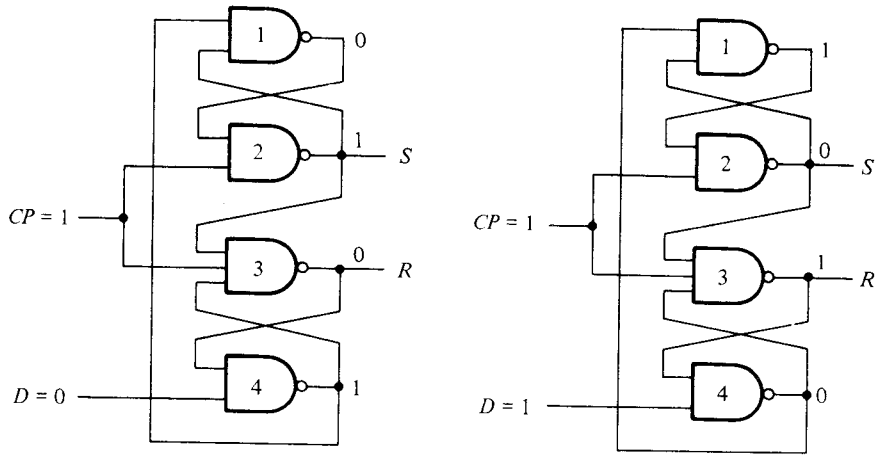


Figura 6-12 Flip-flop tipo D disparado por flanco positivo



(a) Con $CP = 0$



(b) Con $CP = 1$

Figura 6-13 Operación de un flip-flop tipo D disparado por flanco

a 0 ó 1. En cualquier caso, un CP de 0 causa que las salidas de las compuertas 2 y 3 vayan a 1, haciendo $S = R = 1$, lo cual constituye la condición para la salida de estado estable. Cuando $D = 0$, la compuerta 4 tiene una salida de 1 lo que causa que la salida de la compuerta 1 vaya a 0. Cuando $D = 1$, la compuerta 4 irá a 0, lo cual causará que la salida de la compuerta 1 vaya a 1. Estas son las dos condiciones posibles cuando con el terminal CP en 0, se habilitan y cambian las salidas del flip-flop sin importar cual es el valor de D.

Hay un tiempo definido, llamado el tiempo de *establecimiento* durante el cual se debe mantener la entrada D a un valor constante antes de la aplicación del pulso. El tiempo de establecimiento es igual al retardo de propagación a través de las compuertas 4 y 1 ya que un cambio en D cau-

sa un cambio en las salidas de esas dos compuertas. Asúmase ahora que D no cambia durante el tiempo de establecimiento y que la entrada CP se torna 1. Esta situación se dibuja en la Figura 6-13(b). Si $D = 0$ cuando CP se convierta en 1, entonces S permanecerá 1 pero R cambiará a 0. Esto causará que la salida del flip-flop Q vaya a 0 (en la Figura 6-12). Si ahora durante $CP = 1$, hay un cambio en la entrada D, la salida de la compuerta 4 permanecerá en 1 (aunque D vaya a 1), ya que una de las entradas de la compuerta viene de R, la cual se ha mantenido en 0. Solamente cuando CP reaparece en 0, la salida de la compuerta 4 puede cambiar; pero entonces ambas R y S se convierten en 1, no permitiendo ningún cambio en la salida del flip-flop. Sin embargo hay un tiempo definido, llamado el tiempo de *sostenimiento*, el cual no puede ser cambiado por la entrada D después de la aplicación de la transición del flanco positivo móvil del pulso. El tiempo de sostenimiento es igual al retardo de propagación de la compuerta 3, ya que se debe tener seguridad que R se convierta en 0 para poder mantener la salida de la compuerta 4 en 1, independientemente del valor de D.

Si $D = 1$ cuando $CP = 1$, entonces S cambia a 0 pero R permanece en 1, lo cual causa que la salida del flip-flop Q vaya a 1. Un cambio en D, mientras $CP = 1$ no altera S y R porque la compuerta 1 se mantiene en 1 por la señal 0 de S. Cuando CP vaya a cero, ambas R y S irán a 1 para prevenir que la salida sufra algunos cambios.

En suma, cuando el pulso del reloj de entrada hace una transición de flanco móvil positivo, el valor de D se trasfiere a Q. Los cambios en D cuando CP se mantiene en un valor estable de 1 no afectarán a Q. Sin embargo, una transición del pulso de flanco negativo no afectará la salida, como tampoco lo hará cuando $CP = 0$. Entonces, los flip-flops disparados por flancos eliminan cualquier problema de realimentación en los circuitos secuenciales de la misma manera que lo hace el flip-flop maestro esclavo. El tiempo de establecimiento y de sostenimiento deben tenerse en consideración al usar este tipo de flip-flop.

Cuando se usan diferentes tipos de flip-flops en el mismo circuito secuencial, se debe estar seguro que todos los flip-flops hacen la transición al mismo tiempo es decir, durante el flanco positivo o el flanco negativo del pulso. Aquellos flip-flops que se comporten opuestamente a la transición de polaridad adoptada, pueden cambiarse fácilmente agregándoles inversores en los relojes de entrada. Un procedimiento alterno es suministrar ambos pulsos positivos y negativos (por medio de un inversor) y luego aplicar los pulsos positivos a los flip-flops que se disparan durante el flanco negativo y los pulsos negativos a los flip-flops que se disparan durante el flanco positivo, o viceversa.

Entradas directas

Los flip-flops disponibles en cápsulas de CI vienen algunas veces con entradas especiales para puesta a uno o cero del flip-flop de manera asincrónica. Estas entradas se llaman de *puesta a uno directa* (direct preset) y de *puesta a cero directa* (direct clear). Ellas afectan el flip-flop en el valor positivo (o negativo) de la señal de entrada sin que sea necesario el

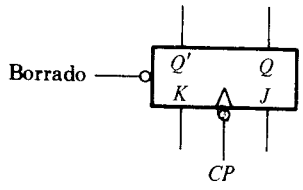


Tabla de función

Entradas				Salidas	
Borrado	Reloj	J	K	Q	Q'
0	X	X	X	0	1
1	↓	0	0	No cambio	
1	↓	0	1	0	1
1	↓	1	0	1	0
1	↓	1	1	Conmuta	

Figura 6-14 Flip-flop *JK* con entrada directa de puesta a cero

pulso de reloj. Estas entradas son útiles para llevar todos los flip-flops a su estado inicial antes de empezar su operación temporizada. Por ejemplo, cuando se suministra potencia por primera vez a un sistema digital el estado de los flip-flops es indeterminado. El interruptor de *puesta a cero* llevará a todos los flip-flops a un estado inicial de cero y el interruptor de *comienzo* (start) empezará la operación de temporizado del sistema. El interruptor de puesta a cero debe "limpiar" todos los flip-flops asincrónicamente sin la necesidad de un pulso.

El símbolo gráfico de un flip-flop maestro esclavo con una entrada de puesta a cero directa se muestra en la Figura 6-14. La entrada de reloj o *CP* tiene un círculo debajo del pequeño triángulo para indicar que las salidas cambian durante la transición negativa del pulso. (La ausencia del pequeño círculo indicaría un flip-flop disparado por flanco positivo). La entrada de puesta a cero directa tiene también un pequeño círculo para indicar que, normalmente, esta entrada debe mantenerse en 1. Si la entrada de puesta a cero se mantiene en 0, el flip-flop permanece en cero independientemente de otras entradas o del pulso de reloj. La tabla de función especifica la operación del circuito. Las X son condición de no importa que indican que un 0 en la entrada directa de puesta a cero inhabilita todas las entradas. Solamente cuando la entrada de puesta a cero es 1 tendría efecto la transición negativa del reloj en las salidas. Las salidas no cambian si $J = K = 0$. El flip-flop conmuta o se complementa cuando $J = K = 1$. Algunos flip-flops pueden tener también una entrada directa de puesta a uno la cual pone la salida Q en (y Q' en 0) asincrónicamente.

Cuando las entradas sincrónicas directas están presentes en un flip-flop maestro esclavo, deben estar conectadas al maestro y al esclavo para poder superponerse a las otras entradas y al reloj. Una entrada directa de puesta a cero en el flip-flop *JK* maestro esclavo de la Figura 6-10 se conecta a las entradas de las compuertas 1, 4 y 8. Una entrada de puesta a cero en el flip-flop *D* de disparo por flanco de la Figura 6-12 se conecta a las entradas de las compuertas 2 y 6.

6-4 ANALISIS DE LOS CIRCUITOS SECUENCIALES TEMPORIZADOS

El comportamiento de los circuitos secuenciales se determina de las entradas, las salidas y los estados de los flip-flops. Ambas entradas y el

siguiente estado son una función de las entradas y el presente estado. El análisis de los circuitos secuenciales consiste en obtener una tabla o un diagrama de la secuencia de tiempo de las entradas, salidas y estados internos. Es posible escribir expresiones de Boole que describan el comportamiento de los circuitos secuenciales. Sin embargo, estas expresiones deben incluir la secuencia de tiempos necesaria directa o indirectamente.

Un diagrama lógico se reconoce como el circuito del circuito secuencial si este incluye flip-flops. Los flip-flops pueden ser de cualquier tipo y el diagrama lógico puede o no incluir compuertas combinacionales. En esta sección, se introduce primero un ejemplo de circuito secuencial temporizado y luego se presentan varios métodos para describir el comportamiento de los circuitos secuenciales. Un ejemplo específico se usará a lo largo de la discusión para ilustrar los diferentes métodos.

Un ejemplo de un circuito secuencial

Un ejemplo de un circuito secuencial temporizado se muestra en la Figura 6-15. Tiene una variable de entrada, una variable de salida y dos flip-flops temporizados RS llamados A y B. Las conexiones realimentadas de las salidas de los flip-flops a las entradas de las compuertas no se muestran en el dibujo para facilitar el trazado del mismo. En vez de ello, se reconocen las conexiones por su letra marcada en cada entrada. Por ejemplo, la entrada marcada x' en la compuerta 1 designa una entrada del complemento de x. La segunda marcada A designa una conexión a la salida normal del flip-flop A.

Se asume que hay disparo por flanco negativo en ambos flip-flops y en la fuente que produce la entrada externa x. Por tanto, las señales para

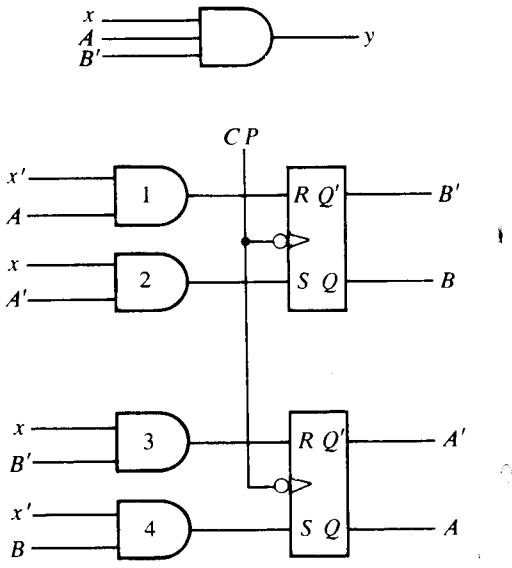


Figura 6-15 Ejemplo de un circuito secuencial temporizado

un estado presente dado están disponibles durante el tiempo en que se determina un pulso de reloj y el siguiente, en cuyo momento el circuito pasa al siguiente estado.

Tabla de estado

La secuencia de tiempo de las entradas, salidas y estados de los flip-flops pueden enumerarse en una *tabla de estado.** La tabla de estado para el circuito de la Figura 6-15 se muestra en la Tabla 6-1. Ella consiste en tres secciones llamadas *estado presente, estado siguiente* y *salida.* El *estado presente* designa los estados de los flip-flops antes de la ocurrencia de un pulso de reloj. El *estado siguiente* muestra los estados de los flip-flops después de la aplicación del pulso de reloj y la sección de *salida* lista los valores de las variables de salida durante el presente estado. Las secciones de estado siguiente y de salida tienen dos columnas, una para $x = 0$ y la otra para $x = 1$.

Tabla 6-1 Tabla de estado para el circuito de la Figura 6-15

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

La deducción de la tabla de estado comienza a partir de un estado inicial asumido. El estado inicial de la mayoría de los circuitos secuenciales prácticos se define como el estado con ceros en todos los flip-flops. Algunos circuitos secuenciales tienen un estado inicial diferente y algunos no tienen ninguno. En cada caso, el análisis puede comenzar a partir de cualquier estado arbitrario. En este ejemplo, se comienza derivando la tabla de estado comenzando con el estado inicial 00.

Cuando el presente estado es 00, $A = 0$ y $B = 0$. Del diagrama lógico, se observa que con los flip-flops en cero y $x = 0$, ninguna de las compuertas AND produce una señal lógica 1. Por tanto, el siguiente estado permanece sin cambiar. Con $AB = 00$ y $x = 1$, la compuerta 2 produce una señal lógica 1 en la entrada S del flip-flop B y la compuerta 3 produce una señal lógica 1 en la entrada R del flip-flop. Cuando un pulso de reloj dispara los flip-flops, A se pone a cero y B se pone a uno, produciendo el siguiente estado 01. Esta información se lista en la primera fila de la tabla de estado.

*Los libros de teoría de los circuitos de conmutación llaman a esta tabla *tabla de transición.* Ellos reservan el nombre *tabla de estado* a una tabla con estados internos representados por símbolos arbitrarios.

De manera similar, se puede deducir el siguiente estado comenzando a partir de los otros tres estados presentes posibles. En general, el siguiente estado es una función de las entradas, el estado presente y el tipo de flip-flop usado. Con flip-flops RS por ejemplo, se debe recordar que un 1 en la entrada S pone en 1 el flip-flop y un 1 en la entrada R lo pone a cero independientemente del estado anterior. Un 0 en ambas entradas S y R deja el flip-flop sin cambio, mientras que un 1 en ambas entradas S y R demostraría un diseño malo y una tabla de estado indeterminada.

Las entradas para la sección de salida son más fáciles de deducir. En este ejemplo, la salida y es igual a 1 solamente cuando $x = 1$, $A = 1$ y $B = 0$. Por tanto, las columnas de salida se marcan con 0, excepto cuando el estado presente es 10 y la entrada $x = 1$, para la cual y se marca con un 1.

La tabla de estado de cualquier circuito secuencial se obtiene por el mismo procedimiento usado en el ejemplo. En general, un circuito secuencial con m flip-flops y n variables de entrada tendrá 2^m filas, una para cada estado. Las secciones del siguiente estado y de salida tendrán cada una 2^n columnas, una para cada combinación de entrada.

Las salidas externas para un circuito secuencial pueden venir de compuertas lógicas o elementos de memoria. La sección de salida en el estado estable es necesaria solamente si hay tres salidas de las compuertas lógicas. Cualquier salida externa tomada directamente de un flip-flop se lista en la columna de presente estado de la tabla de estado. Por tanto la sección de salida de la tabla de estado puede ser excluida si no hay salidas externas de las compuertas lógicas.

Diagrama de estado

La información disponible en la tabla de estado puede representarse gráficamente en un *diagrama de estado*. En este diagrama se representa un estado por un círculo y la transición entre estados se indica por líneas dirigidas que conectan los círculos. El diagrama de estado del circuito secuencial de la Figura 6-15 se muestra en la Figura 6-16. El número binario dentro de cada circuito identifica el estado representado por el

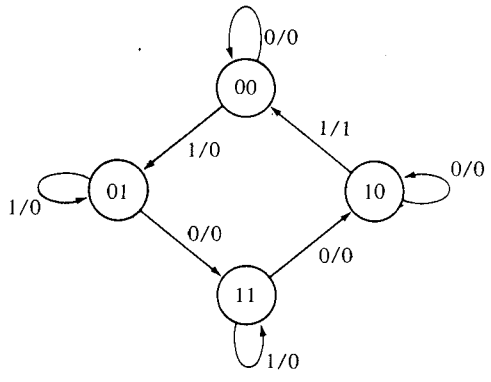


Figura 6-16 Diagrama de estado para el circuito de la Figura 6-15

círculo. Las líneas dirigidas se marcan con dos números binarios separados por /. El valor de entrada que causa la transición de estado se marca primero; el número en seguida del símbolo / da el valor de la salida durante el presente estado. Por ejemplo, la línea dirigida del estado 00 a 01 marcada 1/0, significa que el circuito secuencial está en el estado presente 00 mientras que $x = 1$ y $y = 0$ y que al finalizar el siguiente pulso de reloj, el circuito va al siguiente estado 01. Una línea dirigida que conecta un círculo a sí mismo indica que no hay cambio de estado. El diagrama de estado suministra la misma información que la tabla de estado y se obtiene directamente de la Tabla 6-1.

No hay diferencia entre una tabla de estado y un diagrama de estado excepto en la forma de la presentación. La tabla de estado es más fácil de deducir a partir de un diagrama de lógica dado y el diagrama de estado se desprende directamente de la tabla de estado. El diagrama de estado da una vista pictórica de las transiciones de estado y está en una forma disponible para interpretación binaria de la operación del circuito. El diagrama de estado se usa a menudo como la especificación de diseño inicial de un circuito secuencial.

Ecuaciones de estado

Una *ecuación de estado* (también conocida como una *ecuación de aplicación*) es una expresión algebraica que especifica las condiciones para la transición de estado de un flip-flop. El lado izquierdo de la ecuación denota el estado siguiente del flip-flop y el lado derecho una función de Boole que especifica las condiciones del presente estado que hacen el siguiente estado igual a 1. Una ecuación de estado es similar en forma a una ecuación característica de un flip-flop, excepto que especifica las condiciones del siguiente estado en términos de las variables de entrada externas y otros valores de los flip-flops. La ecuación de estado se deriva directamente de la tabla de estado. Por ejemplo, la ecuación de estado para un flip-flop A se deriva por inspección de la Tabla 6-1. De las siguientes columnas de estado, se nota que el flip-flop A va al estado 1 cuatro veces: cuando $x = 0$ y $AB = 01$ ó 10 ú 11, o cuando $x = 1$ y $AB = 11$. Esto puede expresarse algebraicamente en la ecuación de estado de la siguiente manera:

$$A(t + 1) = (A'B + AB' + AB)x' + ABx$$

El lado derecho de la ecuación de estado es una función de Boole para un *estado presente*. Cuando esta función es igual a 1, la ocurrencia de los pulsos de reloj causa que el flip-flop A tenga el siguiente estado de 1. Cuando una función es igual a 0, el pulso de reloj causará que A tenga el siguiente estado de 0. El lado izquierdo de la ecuación identifica los flip-flops por un símbolo de letra seguido de una designación en función de tiempo $(t + 1)$, para enfatizar que este valor sea alcanzado por el flip-flop, un pulso posterior de la secuencia.

La ecuación de estado es una función de Boole con un tiempo incluido. Es aplicable solamente en los circuitos secuenciales de reloj, ya que $A(t + 1)$ se define para que cambie de valor con la ocurrencia del pulso de reloj en instantes discretos de tiempo.

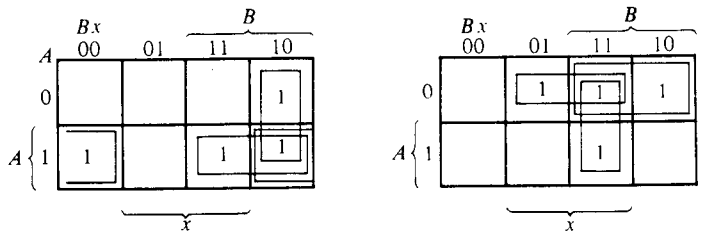


Figura 6-17 Ecuaciones de estado para los flip-flops A y B

La ecuación de estado de un flip-flop A se simplifica por medio de un mapa como se muestra en la Figura 6-17(a). Con alguna manipulación algebraica, la función puede expresarse de la siguiente forma:

$$A(t + 1) = Bx' + (B'x)'A$$

Si se deja que $Bx' = S$ y $B'x = R$, se obtiene la siguiente relación:

$$A(t + 1) = S + R'A$$

la cual es una ecuación característica de un flip-flop RS [Figura 6-4(d)]. Esta relación entre la ecuación de estado y las ecuaciones características del flip-flop puede justificarse por inspección del diagrama lógico de la Figura 6-15. En este se ve que la entrada S del flip-flop A es igual a la función de Boole Bx' y la entrada R es igual a $B'x$. Sustituyendo estas funciones en la ecuación característica del flip-flop, dará como resultado la ecuación de estado para este circuito secuencial.

La ecuación de estado para un flip-flop en un circuito secuencial puede deducirse de una tabla de estado o de un diagrama lógico. La deducción de una tabla de estado consiste en obtener la función de Boole especificando las condiciones que hacen el siguiente estado del flip-flop un 1. La deducción a partir de un diagrama lógico consiste en obtener las funciones de las entradas del flip-flop y sustituirlas en la ecuación característica de la misma.

La derivación de la ecuación de estado del flip-flop B a partir de una tabla de verdad se muestra en el mapa de la Figura 6-17(b). Los 1 marcados en el mapa son las entradas presentes y las combinaciones de entrada que causan que el flip-flop vaya al siguiente estado de 1. Estas condiciones se obtienen directamente de la Tabla 6-1. La forma simplificada que se obtiene en el mapa se manipula algebraicamente y la ecuación de estado que se obtiene es:

$$B(t + 1) = A'x + (Ax')'B$$

La ecuación de estado puede derivarse directamente a partir del diagrama lógico. De la Figura 6-15 se observa que la señal para la entrada S del flip-flop B se genera por la función $A'x$ y la señal para la entrada R

por la función Ax'. Sustituyendo $S = A'x$ y $R = Ax'$ en la ecuación característica del flip-flop RS dada por:

$$B(t + 1) = S + R'B$$

se obtiene la ecuación de estado derivada anteriormente.

Las ecuaciones de estado de todos los flip-flops, conjuntamente con las funciones de salida, especifican totalmente un circuito secuencial. Ellas representan, algebraicamente, la misma información que representa una tabla de estado en forma tabular y un diagrama de estado representa una forma gráfica.

Funciones de entrada de un flip-flop

El diagrama lógico de un circuito secuencial consiste en elementos de memoria y compuertas. La clase de flip-flops y la tabla característica especifican las propiedades lógicas de los elementos de memoria. Las interconexiones entre las compuertas forman un circuito combinacional y se pueden expresar algebraicamente con funciones de Boole. Así, un conocimiento del tipo de flip-flops y una lista de las funciones de Boole del circuito combinacional darán toda la información necesaria para dibujar el diagrama lógico de un circuito secuencial. La parte del circuito combinacional que genera las salidas externas se describe algebraicamente por las *funciones de salida del circuito*. La parte del circuito que genera las entradas de los flip-flops se describe algebraicamente por un conjunto de funciones de Boole llamadas *funciones de entrada del flip-flop* o algunas veces *ecuaciones de entrada*.

Se adoptará la convención de usar dos letras para designar una variable de entrada de un flip-flop: la primera designa el nombre de las entradas y la segunda el nombre del flip-flop. Como un ejemplo, considérese las siguientes funciones de entrada de un flip-flop:

$$JA = BC'x + B'Cx'$$
$$KA = B + y$$

JA y KA designan las variables de Boole. La primera letra en cada una denota la entrada J y K respectivamente del flip-flop JK. La segunda letra A es el símbolo nombre del flip-flop. El lado derecho de cada ecuación es una función de Boole para la correspondiente variable de entrada del flip-flop. La configuración de las dos funciones de entrada se muestra en el diagrama lógico de la Figura 6-18. El flip-flop JK tiene un símbolo de salida A y dos entradas marcadas J y K. El circuito combinacional dibujado en el diagrama es la configuración de una expresión algebraica dada por las funciones de entrada. Las salidas del circuito combinacional se designan por JA y KA en las funciones de salida y van a las entradas J y K del flip-flop A.

De este ejemplo, se observa que la función de entrada del flip-flop es una expresión algebraica para un circuito combinacional. La designación de dos letras es el nombre de una variable para una *salida* de un circuito combinacional. Esta *salida* se conecta siempre a la *entrada* (designada por la primera letra) del flip-flop (designado por la segunda letra).

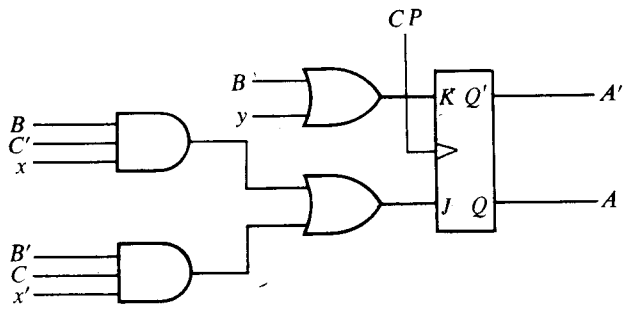


Figura 6-18 Configuración de las funciones de entrada de un flip-flop $JA = BC'x + B'Cx'$ y $KA = B + y$

El circuito secuencial de la Figura 6-15 tiene una entrada x, una entrada y y dos flip-flops RS denotados por A y B. El diagrama lógico puede ser expresado algebraicamente con cuatro funciones de entrada del flip-flop y una función de salida del circuito como sigue:

$$SA = Bx' \qquad RA = B'x$$
$$SB = A'x \qquad RB = Ax'$$
$$y = AB'x$$

Este conjunto de funciones de Boole expecifica totalmente el diagrama lógico. Las variables SA y RA especifican el flip-flop RS llamado A; las variables SB y RB especifican un segundo flip-flop RS denotado por B. La variable y denota la salida. Las expresiones de Boole para las variables especifican parte del circuito combinacional del circuito secuencial.

Las funciones de entrada del flip-flop constituyen una forma algebraica conveniente para especificar un diagrama lógico de un circuito secuencial. Ellas implican el tipo de flip-flop a partir de la primera letra de la variable de entrada y especifican completamente el circuito combinacional que maneja el flip-flop. El tiempo no se incluye explícitamente en estas ecuaciones pero está comprendido a partir de la operación del pulso de reloj. Es conveniente algunas veces especificar algebraicamente un circuito secuencial con funciones de salida del circuito y funciones de entrada del flip-flop en vez de dibujar el diagrama lógico.

6-5 REDUCCION DE ESTADOS Y ASIGNACION*

El análisis de los circuitos secuenciales comienza de un diagrama de circuito y culminan en una tabla de estado o diagrama. El diseño de un circuito secuencial parte de una serie de especificaciones y culmina en un diagrama lógico. Los procedimientos de diseño se presentan comenzando por la Sección 6-7. Esta sección incluye ciertas propiedades de los circuitos secuenciales que pueden ser usados para reducir el número de compuertas y flip-flops durante el diseño.

*Esta sección se puede omitir sin perder continuidad.

Reducción de estado

Cualquier procedimiento de diseño debe considerar el problema de minimizar el costo del circuito final. Las dos reducciones de costo más obvias son las reducciones en el número de flip-flops y el número de compuertas. Debido a que éstos dos ítems son los más obvios, se han estudiado e investigado extensamente. De hecho, una gran porción del objetivo de la teoría de conmutación trata la manera de buscar algoritmos para minimizar el número de flip-flops y compuertas en los circuitos secuenciales.

La reducción del número de flip-flops en un circuito secuencial se conoce como *la reducción de estado* del problema. Los algoritmos de reducción de estado tratan con los procedimientos para reducir el número de estados en la tabla de estado mientras mantiene los requerimientos de entrada-salida externos sin cambio. Como m flip-flops producen 2^m estados, una reducción en el número de estados podría (o no podría) resultar en una reducción en el número de flip-flops. Un efecto impredecible en la reducción del número de flip-flops es que algunas veces el circuito equivalente (con menos flip-flops) podría requerir más compuertas combinacionales.

Se demostrará la necesidad de reducción de estado con un ejemplo. Se comienza con un circuito secuencial cuya especificación se da en el diagrama de estado de la Figura 6-19. En este ejemplo, solamente las secuencias de entrada-salida son importantes; los estados internos se usan solamente para suministrar las secuencias requeridas. Por esta razón, los estados marcados dentro de los círculos se denotan por símbolos de letras en vez de sus valores binarios. Esto es en contraste a un contador binario, donde la secuencia de valores binarios de los estados en sí mismos se toman como salidas.

Hay un número infinito de secuencias de entrada que puede ser aplicado al circuito; cada uno dará como resultado una secuencia única de salida. Como ejemplo, considérese la secuencia de entrada 01010110100 empezando por el estado inicial a. Cada entrada de 0 ó 1 produce una sa-

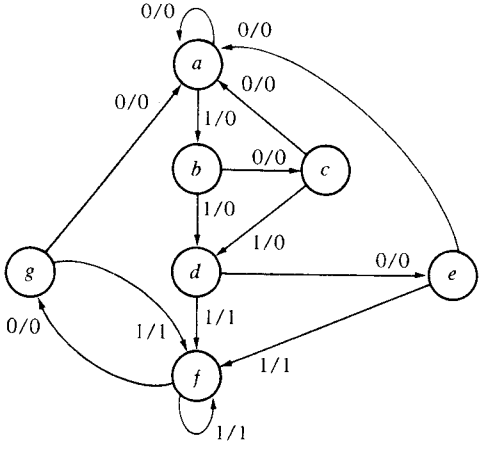


Figura 6-19 Diagrama de estado

lida de 0 ó 1 y causa que el circuito vaya al siguiente estado. De este diagrama de estado, se obtiene la salida y secuencia de estado para una secuencia dada de entrada como sigue: con el circuito en el estado inicial a, una entrada de 0 produce una salida de 0 y el circuito permanece en el estado a. Con el estado presente a y una entrada de 1, la salida es 0 y el siguiente estado es b. Con el estado presente b y una entrada de 0, la salida es 0 y el siguiente estado es c. Continuando este proceso, se encontrará que la secuencia completa es como sigue:

estado	a	a	b	c	d	e	f	f	g	f	g	a
entrada	0	1	0	1	0	1	1	0	1	0	0	
salida	0	0	0	0	0	1	1	0	1	0	0	

En cada columna, se tiene el estado presente, el valor de la entrada y el valor de la salida. El siguiente estado se escribe encima de la siguiente columna. Es importante tener en cuenta que en este circuito los estados en sí mismos son de importancia secundaria porque el interés primordial son las secuencias de salida causadas por las secuencias de entrada.

Asúmase ahora que se tiene un circuito secuencial cuyo diagrama de estado tiene menos de siete estados y se desea compararlo con el circuito cuyo diagrama de estado se da en la Figura 6-19. Si se aplican secuencias de entrada directas a los dos circuitos y ocurren salidas idénticas para todas las secuencias de entrada, entonces se dice que los dos circuitos son equivalentes (en lo que se refiere a la entrada-salida) y se pueden remplazar entre sí. El problema de la reducción de estado es encontrar maneras de reducir el número de estados en un circuito secuencial sin alterar las relaciones de entrada-salida.

Se procederá a reducir el número de estados de este ejemplo. Primero, se necesita una tabla de estado; es más conveniente aplicar los procedimientos para la reducción de estados aquí que en los diagramas de estado. La tabla de estado del circuito se lista en la Tabla 6-2 y se obtiene directamente del diagrama de estado de la Figura 6-19.

Tabla 6-2 Tabla de estado

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Un algoritmo para la reducción de estado de una tabla de estado especificada completamente se da aquí sin prueba alguna: "Se dice que dos estados son equivalentes si, por cada miembro del conjunto de entradas, ellos dan exactamente la misma salida y envían al circuito al mismo estado o a un estado equivalente. Cuando dos estados son equivalentes, uno de ellos puede quitarse sin alterar las relaciones de entrada-salida".

Se aplicará este algoritmo a la Tabla 6-2. Observando la tabla de verdad, se escogen los estados presentes que van al estado siguiente y que tienen la misma salida para ambas combinaciones de entrada. Los estados g y e son dos de tales estados; ellos van a los estados a y f y tienen las salidas de 0 y 1 para $x = 0$ y $x = 1$ respectivamente. Por tanto, los estados g y e son equivalentes y se puede eliminar uno. El procedimiento para quitar un estado y de remplazarlo por un equivalente se demuestra en la Tabla 6-3. La fila con el estado presente g se tacha y el estado g se remplaza por el estado e cada vez que aparezca en las siguientes columnas de estado.

Tabla 6-3 Reduciendo la tabla de estado

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	$f d$	0	1
e	a	$f d$	0	1
f	$g e$	f	0	1
g	a	f	0	1

El estado presente f tiene ahora las entradas siguientes e y f y las salidas 0 y 1 para $x = 0$ y $x = 1$ respectivamente. Los mismos estados siguientes y las salidas aparecen en la fila con el estado presente d. Por tanto, las entradas f y d son equivalentes, el estado f puede quitarse y remplazarse por d. La tabla reducida final se muestra en la Tabla 6-4. El diagrama de estado para la tabla reducida consiste en solamente cinco estados y se muestra en la Figura 6-20. Este diagrama de estado satisface las especializaciones originadas de entrada-salida y producirá la secuencia de salida requerida para una secuencia dada de entrada. La siguiente lista deducida del diagrama de estado de la Figura 6-20 es para la secuencia de entrada usada previamente. Se nota que resulta la misma secuencia de salida aunque la secuencia de estado es diferente:

estado	a	a	b	c	d	e	d	d	e	d	e	a
entrada	0	1	0	1	0	1	1	0	1	0	0	
salida	0	0	0	0	0	1	1	0	1	0	0	

De hecho, esta secuencia es exactamente la misma que se obtuvo de la Figura 6-19, si se remplaza e por g y d por f.

Tabla 6-4 Tabla de estado reducida

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

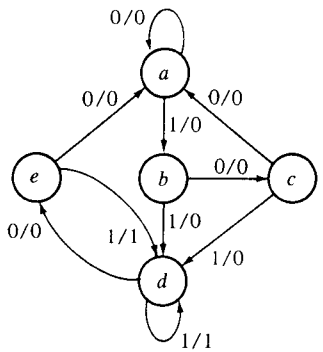


Figura 6-20 Diagrama de estado reducido

De cualquier forma, la reducción de siete a cinco estados no reduce el número de flip-flops. En general, la reducción del número de estados de una tabla de estado se espera que resulte en un circuito con menos equipo. Sin embargo, el hecho de que una tabla de estado haya sido reducida a menos estados no garantiza un ahorro en el número de flip-flops o el número de compuertas.

Vale la pena notar que la reducción en el número de estados de un circuito secuencial es posible si se interesa solamente en las relaciones externas de entrada-salida. Cuando las salidas externas se toman directamente de los flip-flops, las salidas deben ser independientes del número de estados de que se apliquen los algoritmos de reducción de estados.

El circuito secuencial de este ejemplo fue reducido de siete a cinco estados. En cada caso, la representación de los estados con componentes físicos requieren que se usen tres flip-flops, porque m flip-flops pueden representar hasta 2^m estados diferentes. Con tres flip-flops, se pueden formular hasta seis estados binarios denotados por los números binarios 000 hasta 111, con cada bit designando el estado de un flip-flop. Si la tabla de estado de la Tabla 6-2 se usa, se deben asignar valores binarios a los siete estados; el estado restante no se usa. Si se usa la tabla de estado

de la Tabla 6-4, solamente cinco estados necesitan asignación binaria y quedarían tres estados sin usar. Los estados sin usar se tratan como condiciones de no importa durante el diseño del circuito. Como las combinaciones de no importa por lo general ayudan a obtener una función de Boole más simple, de manera parecida el circuito con cinco estados necesitará menos compuertas combinacionales que aquella con siete estados. De cualquier forma, la reducción de siete a cinco estados no reduce el número de flip-flops. En general, la reducción del número de estados de una tabla de estado se espera que resulte en un circuito con menos equipo. Sin embargo, el hecho de que una tabla de estado haya sido reducida a menos estados no garantiza un ahorro en el número de flip-flops o el número de compuertas.

Asignación de estado

El costo de la parte de circuito combinacional de un circuito secuencial puede reducirse usando los métodos de simplificación conocidos para los circuitos combinacionales. Sin embargo, hay otro factor, conocido como el problema de *asignación de estado*, que entra en juego para la minimización de las compuertas combinacionales. Los procedimientos de asignación de estado tienen que ver con los métodos para la asignación de valores binarios o estados de tal forma que se reduce el costo de los circuitos combinacionales que accionan los flip-flops. Esto es particularmente útil cuando se observa un circuito secuencial a partir de sus terminales externos de entrada-salida. Tal circuito puede seguir una secuencia de estados internos, pero los valores binarios de los estados individuales podrían no tener ninguna consecuencia todo el tiempo en que el circuito produzca la secuencia seguida de salidas para una secuencia dada de entradas. Esto no se aplica a los circuitos cuyas salidas externas se toman directamente de los flip-flops con secuencias binarias totalmente especificadas.

Las alternativas de asignación de estado binario disponibles pueden ser demostradas conjuntamente con el circuito secuencial especificado en la Tabla 6-4. Recuérdese que, en este ejemplo, los valores binarios de los estados son inmateriales durante el tiempo en que su secuencia mantenga las relaciones de entrada-salida adecuadas. Por esta razón, cualquier asignación de número binario es satisfactoria siempre que a cada estado se le asigne un número. Tres ejemplos de asignaciones binarias posibles se muestran en la Tabla 6-5 para los cinco estados de la tabla reducida. La asignación 1 es una asignación binaria directa para la secuencia de estados desde *a* hasta *e*. Las otras dos asignaciones se escogen arbitrariamente. De hecho, hay 140 asignaciones diferentes para este circuito (11).

La Tabla 6-6 es la tabla de estado reducida con la asignación binaria 1 sustituida por las letras de los cinco estados.* Es obvio que una asignación binaria diferente resultará en una tabla de estado con valores binarios diferentes para los estados, mientras que las selecciones de entrada-salida permanecen iguales. La forma binaria de la tabla de estado se usa para deducir la parte del circuito combinacional del circuito se-

*Una tabla de estado con asignación binaria se llama algunas veces *tabla de transición*.

Tabla 6-5 Tres asignaciones binarias de estado posibles

Estado	Asignación 1	Asignación 2	Asignación 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

Tabla 6-6 Tabla de estado reducido con asignación binaria 1

	Estado siguiente		Salida	
Estado presente	$x = 0$	$x = 1$	$x = 0$	$x = 1$
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

cuencial. La complejidad del circuito combinacional obtenido, depende de la asignación del estado binario escogido. El diseño del circuito secuencial presentado en esta sección se completa en el Ejemplo 6-1 de la Sección 6-7.

Varios procedimientos se han sugerido para llevar a una asignación binaria particular entre las muchas disponibles. El criterio más común es que la asignación escogida debe resultar en un circuito combinacional simple para las entradas del flip-flop. Sin embargo, hasta el momento, no hay procedimientos de asignación de estado que garanticen un costo mínimo de un circuito combinacional. La asignación de estado es uno de los problemas desafiantes de la teoría de conmutación. El lector interesado puede encontrar mucha literatura completa y creciente de este tópico. Las técnicas para tratar con el problema de asignación de estado se salen del objetivo de este libro.

6-6 TABLAS DE EXCITACION DE LOS FLIP-FLOPS

Las tablas características para varios flip-flops fueron presentadas en la Sección 6-2. Una tabla característica define la propiedad lógica del flip-flop y caracteriza completamente su operación. Los flip-flops de circuito integrado se definen algunas veces por una tabla característica tabulada de manera diferente. Esta segunda forma de las tablas características para los flip-flops *RS*, *JK*, *D* y *T* se muestran en la Tabla 6-7. Ellas representan la misma información que las tablas características de las Figuras 6-4(c) hasta 6-7(c).

La Tabla 6-7(c) define el estado de cada flip-flop como función de sus entradas y su estado previo. $Q(t)$ se refiere al presente estado y $Q(t+1)$

Tabla 6-7 Tablas características del flip-flop

S	R	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	?

(a) RS

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

(b) JK

D	$Q(t+1)$
0	0
1	1

(c) D

T	$Q(t+1)$
0	Q(t)
1	Q'(t)

(d) T

al estado siguiente después de la ocurrencia de un pulso de reloj. La tabla característica del flip-flop RS muestra que el siguiente estado es igual al presente estado cuando las entradas S y R son ambas 0. Cuando la entrada R es igual a 1, el siguiente pulso de reloj pone a cero el flip-flop. Cuando la entrada S es igual a 1 el siguiente pulso de reloj pone a 1 el flip-flop. La interrogación para el siguiente estado cuando ambos S y R sean iguales a 1 designa simultáneamente un estado siguiente indeterminado.

La tabla del flip-flop JK es la misma que la del RS cuando se remplaza J y K por S y R respectivamente, excepto en el caso indeterminado. Cuando J y K son ambos iguales a 1, el estado siguiente es igual al complemento del presente estado, es decir, $Q(t+1) = Q'(t)$. El siguiente estado del flip-flop D es completamente dependiente de la entrada D e independiente del estado presente. El siguiente estado del flip-flop T es el mismo que el estado presente si $T = 0$ y complementando si $T = 1$.

La tabla característica es útil para el análisis y la definición de la operación del flip-flop. Esta especifica el estado siguiente cuando las entradas y el estado presente se conocen. Durante el proceso de diseño se conoce por lo general la transición del presente estado al siguiente y se desea encontrar las condiciones de entrada del flip-flop que causen la transición requerida. Por esta razón, se necesita una tabla que liste las entradas necesarias para un cambio de estado dado. Tal lista se llama una *tabla de excitación*.

La Tabla 6-8 presenta las tablas de excitación de los cuatro flip-flops. Cada tabla consiste en dos columnas, $Q(t)$ y $Q(t+1)$, y una columna para cada entrada para mostrar cómo se logra la transición requerida. Hay cuatro transiciones posibles del presente estado al siguiente. Las condiciones de entrada requeridas para cada una de las cuatro transiciones se derivan de la información disponible en la tabla característica. El símbolo X en las tablas representa la condición de no importa, es decir, no importa que la entrada sea 1 ó 0.

Tabla 6-8 Tablas de excitación de los flip-flops

$Q(t)$	$Q(t+1)$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(a) RS

$Q(t)$	$Q(t+1)$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b) JK

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

(c) D

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

(d) T

Flip-flop *RS*

La tabla de excitación del flip-flop RS se muestra en la Tabla 6-8(a). La primera fila muestra el flip-flop en el estado 0 en el tiempo t. Se desea dejarlo en el estado 0 después de la ocurrencia del pulso. De la tabla característica, se encuentra que si S y R son ambos 0, el flip-flop no cambiará estado. Por tanto, ambas entradas S y R deben ser 0. Sin embargo, no importa si se hace R un 1 cuando ocurre el pulso, ya que resulta dejando el flip-flop en el estado 0. Así, R puede ser 1 ó 0 y el flip-flop permanecerá en el estado 0 en $t+1$. Por tanto, la entrada debajo de R se marca por la condición X de no importa.

Si el flip-flop está en el estado 0 y se desea que vaya al estado 1, entonces a partir de la tabla característica, se encuentra que la única forma de hacer $Q(t+1)$ igual a 1 es hacer $S=1$ y $R=0$. Si el flip-flop va a tener una transición del estado 1 al estado 0 se debe tener $S=0$ y $R=1$.

La última condición que puede ocurrir en un flip-flop es estar en el estado 1 y permanecer en ese mismo estado. Ciertamente R debe ser 0 ya que no se requiere poner a 0 el flip-flop. Sin embargo S debe ser 0 ó 1. Si es 0, el flip-flop no cambia y permanece en el estado 1; si es 1 se llevará el flip-flop al estado 1 como se desea. Así, S se lista como una condición de no importa.

El flip-flop *JK*

La tabla de excitación para el flip-flop JK se muestra en la Tabla 6-8(b). Cuando ambos estado presente y estado siguiente sean 0, la entrada J debe permanecer en 0 y la entrada K puede ser 0 ó 1. Similarmente cuando

el estado presente y siguiente sean 1, la entrada K debe permanecer en 0 mientras que la entrada J puede ser 0 ó 1. Si el flip-flop va a tener una transición del estado 0 al estado 1, J debe ser igual a 1 ya que la entrada J pone a 1 el flip-flop. Sin embargo, la entrada K puede ser 0 ó 1. Si $K = 0$, la condición $J = 1$ pone a uno el flip-flop como se requiere; si $K = 1$ y $J = 1$, el flip-flop se complementa y va del estado 0 al estado 1 como se requiere. De esta manera la entrada K se marca con una condición de no importa para la transición de 0 a 1. Para una transición del estado 1 al estado 0, se debe tener $K = 1$ ya que la entrada K pone a 0 el flip-flop. Pero, la entrada J puede ser 0 ó 1, como $J = 0$ no tiene efecto, y $J = 1$ conjuntamente con $K = 1$ complementa el flip-flop con una transición resultante del estado 1 al estado 0.

La tabla de excitación del flip-flop JK ilustra la ventaja de usar este tipo al diseñar los circuitos secuenciales. El hecho de que tiene tantas condiciones de no importa indica que los circuitos combinacionales para las funciones de entrada deben ser más simples debido a que las funciones de no importa simplifican usualmente la función.

Flip-flop *D*

La tabla de excitación para un flip-flop tipo D se muestra en la Tabla 6-8(c). De la tabla característica, Tabla 6-7(c), se nota que el siguiente estado es siempre igual a la entrada D e independiente del estado presente. Por tanto, D debe ser 0 si $Q(t+1)$ tiene que ser 0, y 1 si $Q(t+1)$ tiene que ser 1, independientemente del valor de $Q(t)$.

Flip-flop *T*

La tabla de excitación para el flip-flop T se muestra en la Tabla 6-8(d). De la tabla característica, Tabla 6-7(d), se encuentra que cuando la entrada $T = 1$ el estado del flip-flop se complementa, cuando $T = 0$ el estado del flip-flop permanece sin cambiar. Por tanto cuando el estado del flip-flop debe permanecer igual, el requerimiento es que $T = 0$. Cuando el estado del flip-flop debe complementarse, T debe ser igual a 1.

Otros flip-flops

El procedimiento de diseño que se va a describir en este capítulo puede ser usado con cualquier flip-flop. Es necesario que se conozca la tabla característica del flip-flop, de la cual es posible desarrollar una nueva tabla de excitación. La tabla de excitación se usa entonces para determinar las funciones de entrada del flip-flop, como se explica en la siguiente sección.

6-7 PROCEDIMIENTO DE DISEÑO

El diseño de un circuito secuencial temporizado comienza a partir de un conjunto de especificaciones y culmina en un diagrama lógico o una lista de funciones de Boole de las cuales se puede obtener el diagrama lógico.

En contraste con el circuito combinacional, el cual está especificado completamente por una tabla de verdad, un circuito secuencial requiere una tabla de verdad para su especificación. El primer paso en el diseño de los circuitos secuenciales es obtener una tabla de estado o una representación equivalente tal como un diagrama de estado o ecuaciones de estado.

Un circuito secuencial sincrónico se hace de flip-flops y compuertas combinacionales. El diseño del circuito consiste en escoger los flip-flops y luego encontrar una estructura de compuertas combinacional, la cual, conjuntamente con los flip-flops, produce un circuito que copa las características enunciadas. El número de flip-flops se determina por el número de estados necesarios en el circuito. El circuito combinacional se deriva de la tabla de estado por los métodos presentados en este capítulo. De hecho, una vez que el tipo y número de los flip-flops se determinen, el proceso de diseño envuelve una trasformación del problema del circuito secuencial al problema del circuito combinacional. De esta manera las técnicas de diseño de los circuitos combinacionales pueden aplicarse.

Esta sección presenta un procedimiento para el diseño de los circuitos secuenciales. Aunque su propósito es servir como guía al principiante, este procedimiento puede acortarse con experiencia. Este procedimiento se minimiza mediante una lista de pasos consecutivos que se recomiendan como sigue:

1. Se establece la descripción en palabras del comportamiento del circuito. Esto puede acompañarse por el diagrama de estado, un diagrama de tiempos, u otra información pertinente.
2. De la información dada del circuito se obtiene la tabla de estado.
3. El número de estados puede reducirse por los métodos de reducción de estados si el circuito secuencial puede caracterizarse por las relaciones de entrada-salida independientes del número de estados.
4. Se asignan valores binarios a cada estado si la tabla de estado obtenida en los pasos 2 ó 3 contienen símbolos de letras.
5. Se determina el número de flip-flops necesarios para asignar una letra a cada una.
6. Se escoge el tipo de flip-flops que se va a usar.
7. A partir de las tablas de estado, se deduce la excitación del circuito y las tablas de salida.
8. Usando un mapa o cualquier otro método de simplificación, se deduce las funciones de salida del circuito y las funciones de entrada del flip-flop.
9. Se dibuja el diagrama lógico.

Las especificaciones en palabras del comportamiento del circuito asumen que el lector está familiarizado con la terminología lógica digital. Es necesario que el diseñador use su intuición y experiencia para llegar a la correcta interpretación de las especificaciones del circuito, porque las descripciones en palabras pueden ser incompletas e inexactas. Sin em-

bargo, una vez que se haya establecido tal especificación y se haya obtenido la tabla de estado, es posible hacer uso del procedimiento formal para diseñar el circuito.

La reducción del número de estados y la asignación de valores binarios a los estados fueron discutidos en la Sección 6-5. En los ejemplos que siguen se asume que el número de estados y su asignación binaria es conocida. Como consecuencia, los pasos 3 y 4 del diseño no se consideran en las discusiones subsecuentes.

Ya se ha mencionado antes que el número de flip-flops se determinan por el número de estados. Un circuito puede tener estados binarios sin usar si el número total de estados es menor que 2^m. Los estados no usados se toman como condiciones de no importa durante el diseño de la parte del circuito combinacional del circuito.

El tipo de flip-flop que se va a usar puede incluirse en las especificaciones del diseño o puede depender en aquello que está disponible al diseñador. Muchos sistemas digitales se construyen totalmente con flip-flops *JK* porque ellos son los más versátiles y disponibles. Cuando hay muchas clases de flip-flops disponibles, es aconsejable usar el flip-flop *RS* o *D* para aplicaciones que requieren trasferencia de datos (tales como registros de desplazamiento). El tipo *T* para aplicaciones que incluyen complementación (tales como contadores binarios), y el tipo *JK* para aplicaciones generales.

La información de salida externa se especifica en la sección de salida de la tabla de estado. De ella podemos deducir las funciones de salida del circuito. La tabla de excitación del circuito es similar a la de los flip-flops individuales, excepto que las condiciones de entrada son dictadas por la información disponible en el presente estado y las columnas del estado siguiente de la tabla de verdad. El método para obtener la tabla de excitación y las funciones simplificadas de entrada del flip-flop es mejor ilustrarlo con un ejemplo.

Se desea diseñar un circuito secuencial temporizado cuyo diagrama de estado se da en la Figura 6-21. El tipo de flip-flop usado es el *JK*.

El diagrama de estado consiste en cuatro estados con valores binarios ya asignados. Como las líneas designadas se marcan con un solo dígito binario sin una /, se concluye que hay una variable de entrada y

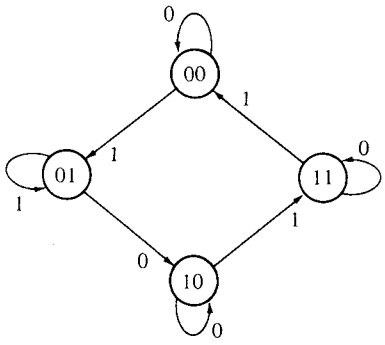


Figura 6-21 Diagrama de estado

ninguna variable de salida. (El estado de los flip-flops puede considerarse como las salidas del circuito.) Los dos flip-flops necesarios para representar los cuatro estados se designan como A y B. La variable de entrada se designa x.

La tabla de estado para este circuito, derivada del diagrama de estado, se muestra en la Tabla 6-9. Nótese que no hay sección de salida para este circuito. Se mostrará ahora el procedimiento para obtener la tabla de excitación y la estructura de la compuerta combinacional.

La derivación de la tabla de excitación se facilitará si se reordena la tabla de estado en forma diferente. Esta forma se muestra en la Tabla 6-10, donde el estado presente y las variables de entrada se reordenan en la forma de tabla de verdad. El valor del estado siguiente para cada estado presente y las condiciones de entrada se copian de la Tabla 6-9. La tabla de excitación del circuito es una lista de condiciones de entrada del flip-flop que causan las transiciones de estado requeridas y es una función del tipo de flip-flop usado. Como este ejemplo especifica flip-flops JK, se necesitan columnas para las entradas J y K del flip-flop A (denotadas por JA y KA) y B (denotadas por JB y KB).

Tabla 6-9 Tabla de estado

Estado presente		Estado siguiente			
		$x = 0$		$x = 1$	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

La tabla de excitación para el flip-flop JK fue derivada en la Tabla 6-8(b). Esta tabla se usa ahora para deducir la tabla de excitación del circuito. Por ejemplo, en la primera fila de la Tabla 6-10 se tiene una transición del flip-flop A de 0 en el presente estado a 0 en el estado siguiente. En la Tabla 6-8(b) se encuentra que los estados de transición de 0 a 0 requieren que la entrada $J = 0$ y la entrada $K = X$. Así 0 y X se copian en la primera fila bajo JA y KA, respectivamente. Como la primera fila muestra también la transición del flip-flop B de 0 en el presente estado a 0 en el siguiente estado, 0 y X se copian en la primera columna bajo JB y KB. La segunda fila de la Tabla 6-10 muestra una transición del flip-flop B de 0 en el presente estado a 1 en el siguiente estado. De la Tabla 6-8(b) se encuentra que una transición de 0 a 1 requiere que la entrada $J = 1$ y la entrada $K = X$. Así 1 y X se copian en la segunda fila bajo JB y KB respectivamente. Este proceso se continúa para cada fila de la tabla de verdad y para cada flip-flop con las condiciones de entrada especificadas

Tabla 6-10 Tabla de excitación

Entradas de los circuitos combinacionales			Siguiente estado		Salidas del circuito combinacional			
Estado presente		Entrada			Entradas de los flip-flops			
A	*B*	*x*	*A*	*B*	*JA*	*KA*	*JB*	*KB*
0	0	0	0	0	0	*X*	0	*X*
0	0	1	0	1	0	*X*	1	*X*
0	1	0	1	0	1	*X*	*X*	1
0	1	1	0	1	0	*X*	*X*	0
1	0	0	1	0	*X*	0	0	*X*
1	0	1	1	1	*X*	0	1	*X*
1	1	0	1	1	*X*	0	*X*	0
1	1	1	0	0	*X*	1	*X*	1

en la Tabla 6-8(b) copiadas en la fila correspondiente del flip-flop particular considerado.

Hágase una pausa y considérese la información disponible en una tabla de excitación tal como la Tabla 6-10. Se sabe que un circuito secuencial consiste en un número de flip-flops y un circuito combinacional. La Figura 6-22 muestra los dos flip-flops *JK* necesarios para el circuito y un rectángulo para representar el circuito combinacional. Es claro del diagrama de bloque que las salidas del circuito combinacional vayan a las entradas de los flip-flops y a las salidas externas (si se especifica). Las entradas del circuito combinacional son las entradas externas y los valores de estado presentes de los flip-flops. Sin embargo, las funciones de Boole que especifican un circuito combinacional se derivan de una tabla de verdad que muestra las relaciones de entrada-salida del circuito. La tabla de verdad que describe el circuito combinacional es disponible en la tabla de excitación. Las *entradas* del circuito combinacional se especifican bajo el presente estado y las columnas de entrada, las *salidas* del circuito combinacional se especifican bajo las columnas de entrada de los flip-flops. Así, una tabla de excitación trasforma un diagrama de estado a la tabla de verdad necesaria para el diseño de la parte del circuito combinacional del circuito secuencial.

Las funciones de Boole simplificadas para el circuito combinacional pueden ahora derivarse. Las entradas son las variables *A*, *B* y *x*; las salidas son las variables *JA*, *KA*, *JB* y *KB*. La información de la tabla de verdad se trasfiere a los mapas de la Figura 6-23, donde se derivan las cuatro funciones simplificadas de la entrada de los flip-flops:

$$JA = Bx' \qquad KA = Bx$$
$$JB = x \qquad KB = A \odot x$$

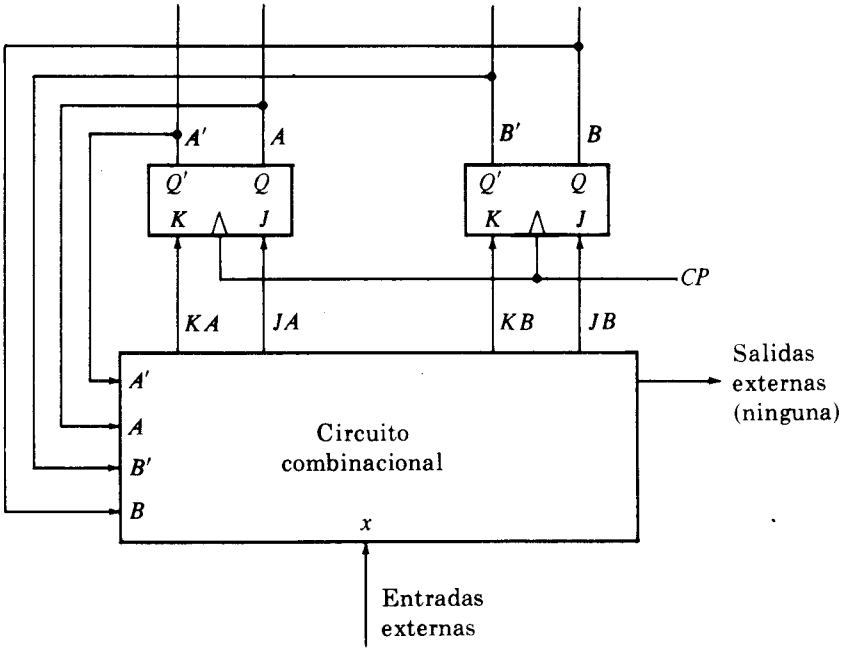


Figura 6-22 Diagrama de bloque del circuito secuencial

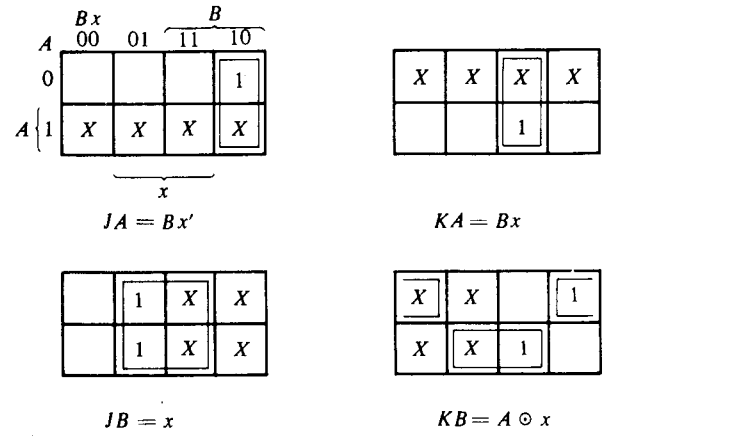


Figura 6-23 Mapas del circuito combinacional

El diagrama lógico se dibuja en la Figura 6-24 y consiste en dos flip-flops, dos compuertas AND, una compuerta de equivalencia y un inversor.

Con alguna experiencia, es posible reducir la cantidad de trabajo envuelto en el diseño del circuito combinacional. Por ejemplo, es posible obtener la información para los mapas de la Figura 6-23 directamente de la Tabla 6-9 sin tener que derivar la Tabla 6-10. Esto se hace repasando sistemáticamente cada estado presente y la combinación de entrada en la

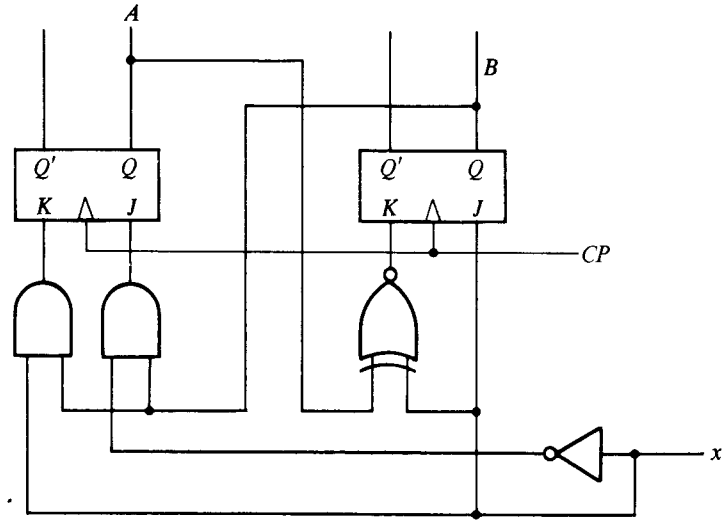


Figura 6-24 Diagrama lógico del circuito secuencial

Tabla 6-9 y comparándola con los valores binarios del siguiente estado correspondiente. Las condiciones de entrada necesarias, como se especifican por la excitación de los flip-flops en la Tabla 6-8, se determinan entonces. En vez de insertar el 0, 1 ó x así obtenidos en la tabla de excitación, se pueden escribir directamente en el cuadrado apropiado del mapa apropiado.

La tabla de excitación de un circuito secuencial con m flip-flops, k entradas por flip-flop y n entradas externas consiste en $m+n$ columnas para el estado presente y las variables de entrada y hasta 2^{m+n} filas listadas en alguna cuenta binaria conveniente. La siguiente sección de estado tiene m columnas, una para cada flip-flop. Los valores de entrada de los flip-flops se listan en mk columnas, una para cada entrada de cada flip-flop. Si el circuito contiene j salidas, la tabla debe incluir j columnas. La tabla de verdad del circuito combinacional se toma de la tabla de excitación considerando el estado presente $m+n$ y las columnas de entrada como *entradas*, y los valores de entrada del flip-flop $mk+j$ y las salidas externas como *salidas*.

Diseño con estados no usados

Un circuito con m flip-flops puede tener 2^m estados. Hay ocasiones cuando un circuito secuencial puede usar menos que este máximo número de estados. Los estados que no se usan en la especificación del circuito secuencial no se listan en la tabla de estado. Cuando se simplifican las funciones de entrada de los flip-flops, los estados sin usar pueden ser tratados como condiciones de no importa.

EJEMPLO 6-1: Completar el diseño del circuito secuencial presentado en la Sección 6-5. Use la tabla de estado reducida con

Tabla 6-11 Tabla de excitación para el Ejemplo 6-1

Estado presente			Entrada	Estado siguiente			Entradas de flip-flops						Salidas
A	*B*	*C*	*x*	*A*	*B*	*C*	*SA*	*RA*	*SB*	*RB*	*SC*	*RC*	*y*
0	0	1	0	0	0	1	0	*X*	0	*X*	*X*	0	0
0	0	1	1	0	1	0	0	*X*	1	0	0	1	0
0	1	0	0	0	1	1	0	*X*	*X*	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	*X*	0
0	1	1	0	0	0	1	0	*X*	0	1	*X*	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	*X*	0	0	*X*	1	0	0
1	0	0	1	1	0	0	*X*	0	0	*X*	0	*X*	1
1	0	1	0	0	0	1	0	1	0	*X*	*X*	0	0
1	0	1	1	1	0	0	*X*	0	0	*X*	0	1	1

la asignación 1 tal como se da en la Tabla 6-6. El circuito debe usar flip-flops *RS*.

La tabla de estado de la Tabla 6-6 se redibuja en la Tabla 6-11 en la forma conveniente para obtener la tabla de excitación. Las condiciones de entrada del flip-flop se deriva de las columnas del estado presente y del siguiente estado de la tabla de estado. Como se usan los flip-flops *RS* es necesario referirse a la Tabla 6-8(a) para las condiciones de excitación de este tipo de flip-flop. A los tres flip-flops se les da los nombres de las variables *A*, *B* y *C*. La variable de entrada es *x* y la variable de salida es *y*. La tabla de excitación del circuito suministra toda la información necesaria para el diseño.

Hay tres estados sin usar en este circuito: los estados binarios 000, 110 y 111. Cuando se incluye una entrada de 0 ó 1 con estos estados no usados se obtienen seis términos mínimos, de no importa: 0, 1, 12, 13, 14 y 15. Estas seis combinaciones binarias no se listan en la tabla de verdad bajo el estado presente y la entrada y se tratan como términos de no importa.

La parte del circuito combinacional del circuito secuencial se simplifica por medio de los mapas de la Figura 6-25. Hay siete mapas en el diagrama, seis mapas son para simplificar las funciones de entrada para los tres flip-flops *RS*. El séptimo mapa es para simplificar la salida *y*. Cada mapa tiene seis *X* en los cuadrados de los términos mínimos de no importa 0, 1, 2, 13, 14 y 15. Los otros términos de no importa en los mapas provienen de las *X* en las columnas de entrada del flip-flop de la tabla. Las funciones simplificadas se listan bajo cada mapa. El diagrama lógico obtenido de estas funciones de Boole se dibujan en la Figura 6-26.

Un factor olvidado hasta este momento en el diseño es el estado inicial del circuito secuencial. Cuando se le da potencia a un sistema digital

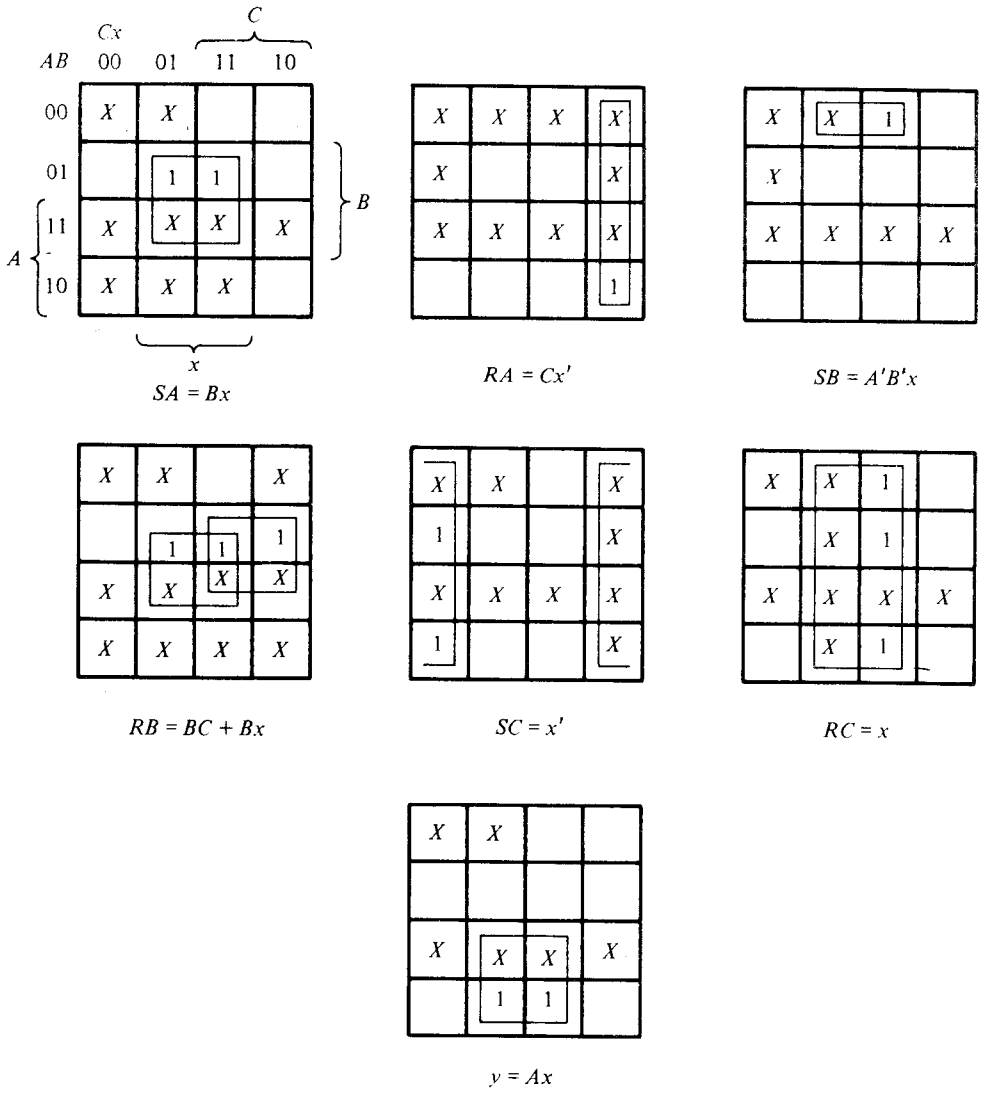


Figura 6-25 Mapas para simplificar el circuito secuencial del Ejemplo 6-1

por primera vez, no se conoce en qué estado se fijará el flip-flop. Es costumbre suministrar una entrada *maestra de puesta a uno* (master-re-set) cuyo propósito es iniciar los estados de todos los flip-flops en el sistema. Típicamente, la maestra de puesta a uno es una señal aplicada a todos los flip-flops asincrónicos antes de comenzar las operaciones temporizadas. En la mayoría de los casos los flip-flops se llevan a 0 por medio de la señal maestra de puesta a 0, pero algunos serán puestos a 1. Por ejemplo, el circuito de la Figura 6-26 puede inicialmente ponerse a 0 con un estado $ABC = 001$, ya que el estado 000 no es un estado válido para este circuito.

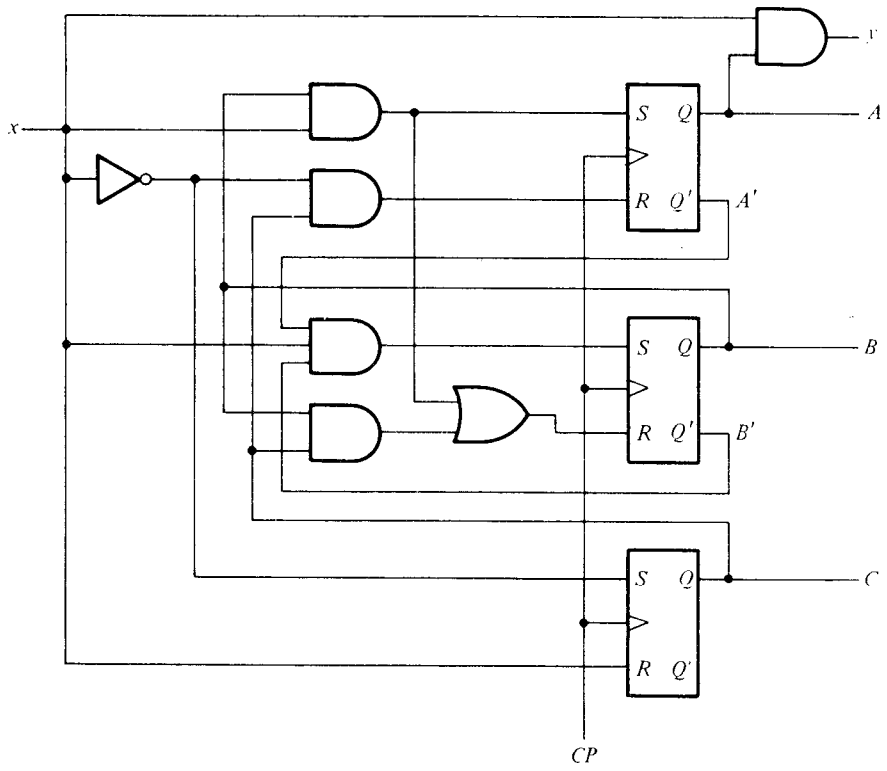


Figura 6-26 Diagrama lógico para el Ejemplo 6-1

¿Pero qué pasa si el circuito no se pone a cero con un estado válido inicial? O lo que es peor, ¿qué pasa si debido a la señal de ruido o cualquier otra razón imprevista, el circuito se encuentra en uno de estos estados inválidos? En este caso es necesario asegurar que el circuito eventualmente vaya a uno de los estados válidos para regresar a la operación normal. De otra manera, si el circuito secuencial circula dentro de los estados inválidos, no habrá manera de llevarlo de nuevo a la secuencia intentada de las transiciones de estado. Aunque se puede asumir que esta condición indeseable supuestamente no ocurre, un diseñador cuidadoso puede prevenir que esta situación nunca ocurra.

Se había expresado previamente que los estados sin usar en un circuito secuencial pueden ser tratados como condiciones de no importa. Una vez que se diseña el circuito, los m flip-flops en el sistema pueden estar en cualquiera de los 2^m estados posibles. Si algunos de estos estados se tomaran como condiciones de no importa, el circuito puede ser investigado para determinar el efecto de estos estados sin usar. El estado siguiente de los estados inválidos pueden determinarse del análisis del circuito. De todas maneras, es siempre acertado analizar un circuito obtenido de un diseño, para asegurar que no se cometan errores durante el proceso.

EJEMPLO 6-2: Analizar el circuito secuencial obtenido en el Ejemplo 6-1 y determinar el efecto de los estados sin usar.

Los estados sin usar son 000, 110 y 111. El análisis del circuito se hace por el método esbozado en la Sección 6-4. Los mapas de la Figura 6-25 pueden ayudar también en el análisis. Lo que se necesita aquí es comenzar con el diagrama del circuito de la Figura 6-26 y derivar la tabla o el diagrama. Si la tabla de estado derivada es idéntica a la Tabla 6-6 (o la parte de la tabla de estado de la Tabla 6-11), entonces se sabe que el diseño es correcto. En suma, se debe determinar los estados siguientes de los estados sin usar 000, 110 y 111.

Los mapas de la Figura 6-25 pueden ayudar a encontrar el siguiente estado de cada una de las entradas sin usar. Tómese, por ejemplo, el estado sin usar 000. Si en este circuito, por alguna razón, se encuentra en el presente estado 000, una entrada $x = 0$ trasferirá a otro (o al mismo) estado siguiente. Se investigará primero el término mínimo $ABCx = 0000$. De los mapas, se ve que este término mínimo no se incluye en ninguna función excepto para SC, es decir, la entrada de puesta a uno del flip-flop C. Por tanto, los flip-flops A y B no cambiarán pero el flip-flop C se pondrá a 1. Como el presente estado es $ABC = 000$, el siguiente estado será $ABC = 001$. Los mapas mostrarán también que el término mínimo $ABCx = 0001$ se incluye en las funciones para SB y RC. Por tanto B se pondrá a uno y C se pondrá a cero. Comenzando con $ABC = 000$ y poniendo a uno a B, se obtiene el siguiente estado $ABC = 010$ (C ya se ha puesto a cero). La investigación del mapa para la salida y demuestra que y será cero para estos dos términos mínimos.

El resultado del procedimiento de análisis se muestra en el diagrama de estado de la Figura 6-27. El circuito opera como se ha diseñado, siempre y cuando esté dentro de los estados 001, 010, 011, 100 y 101. Si alguna vez se encuentra en uno de los estados

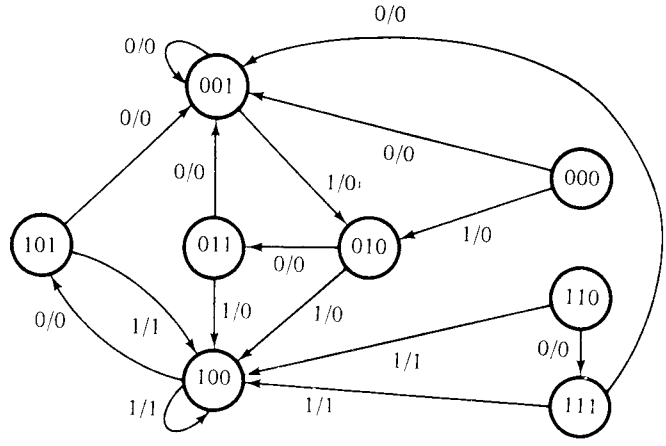


Figura 6-27 Diagrama de estado para el circuito de la Figura 6-26

inválidos 000, 110 ó 111, irá a alguno de los estados válidos en unc de los dos pulsos siguientes. El circuito será así de autocomienzo y autocorrección ya que eventualmente irá a un estado válido a partir del cual continuará operando de acuerdo a lo requerido.

Una situación indeseable hubiera ocurrido si el estado siguiente de 110 para $x = 1$ hubiera sido 111 y el estado siguiente de 111 para $x = 0$, 110. Entonces, si el circuito comienza de 110 ó 111, circulará y se mantendrá entre estos dos estados para siempre. Los estados no usados que causan tal comportamiento indeseable deben ser evitados; si se detecta su existencia, el circuito debe ser rediseñado. Esto puede hacerse más fácilmente especificando un estado siguiente válido para cualquier estado sin usar que se haya encontrado circulando entre estados inválidos.

6-8 DISEÑO DE CONTADORES

Un circuito secuencial que pasa por una secuencia preestablecida de estados después de la aplicación de pulsos se llama un *contador*. Los pulsos de entrada, llamados *pulsos de cuenta*, pueden ser pulsos de reloj, o ellos pueden originarse en una fuente externa y pueden ocurrir en intervalos establecidos de tiempo o aleatoriamente. En un contador, la secuencia de estados puede seguir una cuenta binaria o cualquier otra secuencia de estados. Los contadores se encuentran en la mayoría de los equipos que contienen lógica digital. Ellos se usan para contar el número de ocurrencias de un evento y se usan para generar secuencias de tiempo para controlar las operaciones en un sistema digital.

De las diferentes secuencias que un contador debe seguir, la secuencia binaria directa es la más simple y la más directa. Un contador que sigue la secuencia binaria se llama *contador binario*. Un contador de n bits consiste en n flip-flops y puede contar en binario de 0 hasta $2^n - 1$. Como un ejemplo, el diagrama de estado de un contador de 3 bits se muestra en la Figura 6-28. Como se ve en los diagramas de estado indicados dentro de los círculos, las salidas de los flip-flops repiten la secuencia de cuenta binaria con un regreso a 000 después de 111. Las líneas dirigidas

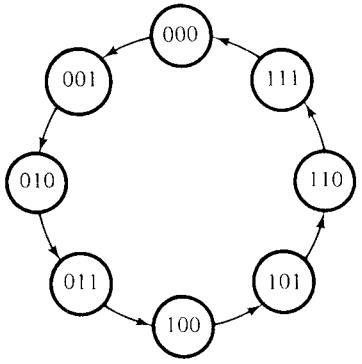


Figura 6-28 Diagrama de estado de un contador binario de 3 bits

entre círculos no se marcan con valores de entrada-salida como en otros diagramas de estado. Recuérdese que las transiciones de estado en dos circuitos secuenciales temporizados ocurren durante un pulso de reloj; los flip-flops permanecen en sus estados presentes si no ocurre ningún pulso. Por esta razón, el pulso de reloj variable CP no aparece explícitamente como una variable de entrada en un diagrama de estado o tabla de estado. Desde este punto de vista, el diagrama de estado de un contador no tiene que mostrar valores de entrada-salida a lo largo de las líneas dirigidas. La única entrada al circuito es el pulso de cuenta, y las salidas se especifican directamente con los estados presentes de los flip-flops. El siguiente estado del contador depende enteramente de su estado presente y la transición de estado ocurre cada vez que ocurre el pulso. Debido a esta propiedad, se especifica completamente un contador por medio de una lista de *secuencia de cuenta,* es decir, la secuencia de los estados binarios que se le suceden.

La secuencia de cuenta de un contador binario de 3 estados se da en la Tabla 6-12. El siguiente número en la secuencia representa el siguiente estado alcanzado por el circuito después de la aplicación del pulso de cuenta. La secuencia de cuenta se repite una vez haya alcanzado el último valor, de tal manera que el estado 000 es el estado siguiente después de 111. La secuencia de cuenta da toda la información necesaria para diseñar el circuito. No es necesario listar los estados siguientes en una columna separada porque se puede leer del número siguiente en la secuencia. El diseño de contadores sigue el mismo procedimiento que aquel esbozado en la Sección 6-7, excepto que la tabla de excitación puede obtenerse directamente de la secuencia de cuenta.

Tabla 6-12 Tabla de excitación para un contador binario de tres bits

Secuencia de cuenta			Entradas del flip-flop		
A_2	A_1	A_0	TA_2	TA_1	TA_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

La Tabla 6-12 es la tabla de excitación para el contador binario de 3 bits. Se les da designaciones de variables A_2, A_1 y A_0 a los tres flip-flops. Los contadores binarios se construyen más eficientemente con flip-flops T (o flip-flops JK con J y K unidas). La excitación del flip-flop para las entradas T se deriva de la tabla de excitación del flip-flop T y por inspección de la transición de estado de una cuenta dada (estado presente)

a la siguiente bajo ella (estado siguiente). Como ilustración, considérese las entradas del flip-flop para la fila 001. El estado presente aquí es 001 y el siguiente es 010, el cual es la siguiente cuenta en la secuencia. Comparando estas dos cuentas, se nota que A_2 va de 0 a 0; y así lo hace TA_2 con un 0 porque el flip-flop A_2 debe permanecer sin cambiar cuando ocurre un pulso de reloj. A_1 va de 0 a 1; y así TA_1 se marca con un 1 porque el flip-flop debe ser complementado en el siguiente pulso de reloj. De manera similar A_0 va de 1 a 0, indicando que esta puede complementarse, y así TA_0 se marca con un 1. La última columna con el estado presente 111 se compara con la primera cuenta 000 la cual es su estado siguiente. Al pasar de todos los unos a todos los ceros, se requiere que todos los tres flip-flops se complementen.

Las funciones de entrada de los flip-flops de las tablas de excitación se simplifican en los mapas de la Figura 6-29. Las funciones de Boole listadas bajo cada mapa especifican la parte de circuito combinacional del contador. Incluyendo estas funciones con los tres flip-flops, se obtiene un diagrama lógico del contador de la manera mostrada en la Figura 6-30.

Un contador con n flip-flops puede tener una secuencia binaria de menos de 2^n números. Un contador BDC cuenta la secuencia binaria desde 0000 hasta 1001 y regresa a 0000 para repetir la secuencia. Otros contadores pueden seguir una secuencia arbitraria, la cual puede no ser la secuencia binaria directa. De todas formas, el procedimiento de diseño es el mismo. La secuencia de cuenta se lista y la tabla de excitación se obtiene comparando una cuenta presente con la siguiente cuenta listada

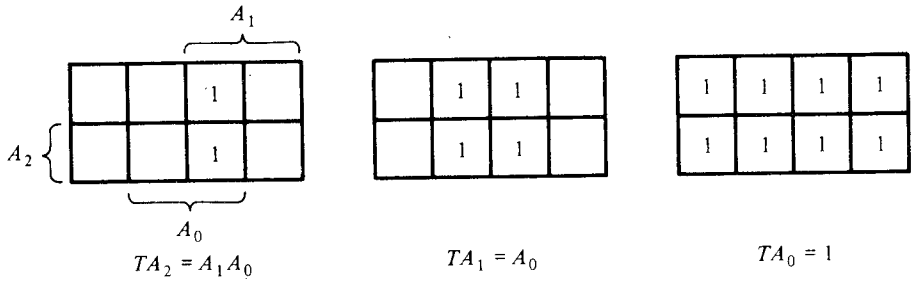


Figura 6-29 Mapas para un contador binario de 3 bits

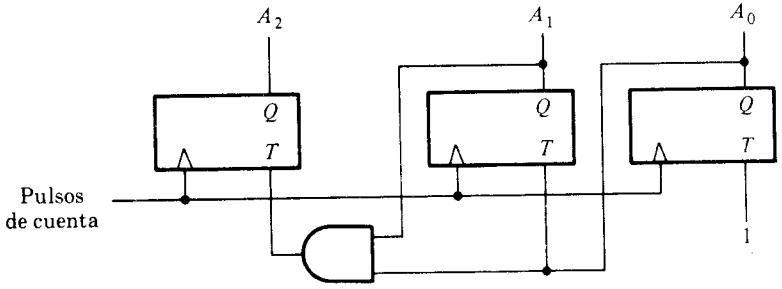


Figura 6-30 Diagrama lógico de un contador binario de 3 bits

bajo ella. Una secuencia de cuenta tabulada siempre asume una cuenta repetida, de tal forma que el estado siguiente de la última entrada es la primera cuenta listada.

EJEMPLO 6-3: Diséñese un contador que tenga una secuencia repetida de seis estados como la listada en la Tabla 6-13.

En esta secuencia, los flip-flops B y C repiten la cuenta binaria 00, 01, 10 mientras que el flip-flop A alterna entre los estados 0 y 1 cada tres cuentas. La secuencia de cuenta para A, B, C no es binaria directa y los dos estados 011 y 111 no se usan. La escogencia de los flip-flops JK resulta en una tabla de excitación de la Tabla 6-13. Las entradas KB y KC tienen solamente 1 y X en

Tabla 6-13 Tabla de excitación para el Ejemplo 6-3

Secuencia de cuenta			Entradas del flip-flop					
A	*B*	*C*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*
0	0	0	0	*X*	0	*X*	1	*X*
0	0	1	0	*X*	1	*X*	*X*	1
0	1	0	1	*X*	*X*	1	0	*X*
1	0	0	*X*	0	0	*X*	1	*X*
1	0	1	*X*	0	1	*X*	*X*	1
1	1	0	*X*	1	*X*	1	0	*X*

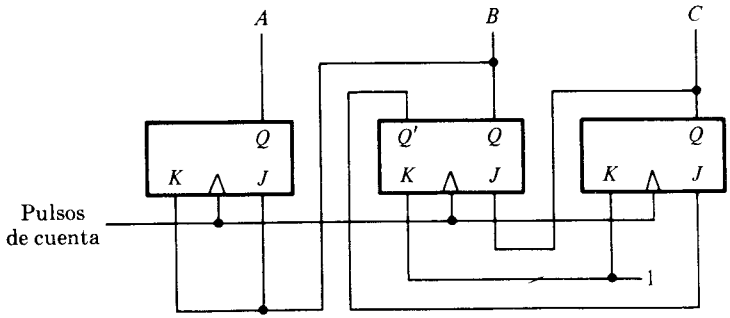
sus columnas, de tal manera que esas entradas sean siempre 1. Las otras funciones de entrada de los flip-flops pueden simplificarse usando términos mínimos 3 y 7 como condiciones de no importa. Las funciones simplificadas son:

$$JA = B \qquad KA = B$$

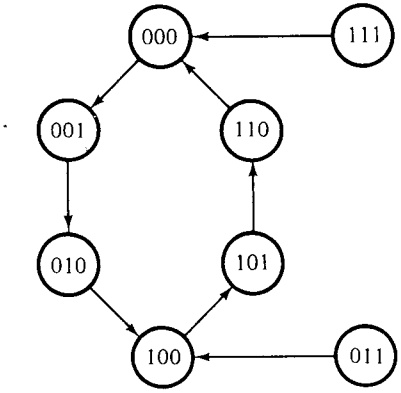
$$JB = C \qquad KB = 1$$

$$JC = B' \qquad KC = 1$$

El diagrama lógico del contador se muestra en la Figura 6-31(a). Como hay dos estados sin usar, se analiza el circuito para determinar su efecto. El diagrama de estado así obtenido se dibuja en la Figura 6-31(b). Si el circuito por algún motivo va a un estado inválido, el siguiente pulso de cuenta lo trasfiere a uno de los estados válidos y continúa contando correctamente. Así, el contador se autoinicia. Un contador autocomenzante es aquel que puede comenzar en cualquier estado y eventualmente alcanzar la secuencia de cuenta normal.



(a) Diagrama lógico del contador



(b) Diagrama de estado del contador

Figura 6-31 Solución al Ejemplo 6-3

6-9 DISEÑO CON ECUACIONES DE ESTADO

Un circuito secuencial puede diseñarse por medio de ecuaciones de estado en vez de una tabla de excitación. Como se muestra en la Sección 6-4, una ecuación de estado es una expresión algebraica que da las condiciones para el siguiente estado como una función del estado presente y las variables de entrada. Las ecuaciones de estado de un circuito secuencial expresan en forma algebraica la misma información la cual es expresada en forma tabular en la tabla de estado.

El método de la ecuación de estado es conveniente cuando el circuito se haya especificado en esta forma de la tabla de estado. Este es el método preferido cuando se usan los flip-flops D. El método puede ser algunas veces conveniente de usar con flip-flops JK. La aplicación de este procedimiento en los circuitos con flip-flops RS o T es posible pero encierra una

cantidad considerable de manipulación algebraica. Aquí se mostrará la aplicación de este método a los circuitos secuenciales usando flip-flops D ó JK. El punto de comienzo en cada caso es la ecuación característica del flip-flop derivado en la Sección 6-2.

Circuitos secuenciales con flip-flops D

La ecuación característica del flip-flop D se deriva en la Figura 6-5(d):

$$Q(t + 1) = D$$

Esta ecuación establece que el siguiente estado del flip-flop es igual al valor presente de su salida D y es independiente del valor del presente estado. Esto significa que las entradas para el siguiente estado en la tabla de estado son exactamente las mismas que las entradas D. Por tanto, no es necesario derivar las condiciones de entrada del flip-flop para la tabla de excitación porque esta información está disponible ya en las columnas del siguiente estado.

Tómese, por ejemplo, la tabla de excitación de la Tabla 6-10. La siguiente columna de estado para A tiene cuatro unos, de la misma manera que la columna para el siguiente estado de B. Para diseñar este circuito con flip-flops D, se escriben las ecuaciones de estado y se forma la ecuación con ellos a las entradas D correspondientes:

$$A(t + 1) = DA(A, B, x) = \Sigma(2, 4, 5, 6)$$
$$B(t + 1) = DB(A, B, x) = \Sigma(1, 3, 5, 6)$$

donde DA y DB son las funciones de entrada de los flip-flops para los flip-flops A y B respectivamente, y cada función se expresa como la suma de los cuatro términos mínimos. Las funciones simplificadas pueden obtenerse por medio de dos mapas de tres variables. Las funciones simplificadas de entrada al flip-flop son:

$$DA = AB' + Bx'$$
$$DB = A'x + B'x + ABx'$$

Si hay estados sin usar en el circuito secuencial, deben considerarse conjuntamente con las entradas como combinaciones de no importa. Los términos mínimos de no importa así obtenidos pueden usarse para simplificar las ecuaciones de estado de las funciones de entrada del flip-flop D.

EJEMPLO 6-4: Diséñese un circuito secuencial con cuatro flip-flops A, B, C y D. Los estados siguientes de B, C y D son iguales a los estados presentes de A, B y C respectivamente. El estado siguiente de A es igual a la OR-exclusiva de los estados presentes de C y D.

A partir del enunciado del problema, es conveniente escribir primero las ecuaciones de estado para el circuito:

$$A(t+1) = C \oplus D$$
$$B(t+1) = A$$
$$C(t+1) = B$$
$$D(t+1) = C$$

Este circuito especifica un *registro de corrimiento por realimentación* (feedback shift register). En este registro, cada flip-flop trasfiere o desplaza su contenido al siguiente flip-flop cuando ocurre un pulso de reloj, pero el siguiente estado del primer flip-flop (A en este caso) es alguna función del estado presente de otros flip-flops. Como las ecuaciones de estado son muy simples, el flip-flop más conveniente de usar es el tipo D.

Las funciones de entrada del flip-flop para este circuito se toman directamente de las ecuaciones de estado, con la siguiente variable de estado remplazada por la variable de entrada del flip-flop:

$$DA = C \oplus D$$
$$DB = A$$
$$DC = B$$
$$DD = C$$

El circuito puede construirse con cuatro flip-flops D y una compuerta OR-exclusiva.

Ecuaciones de estado con flip-flops *JK**

La ecuación característica para el flip-flop JK se deriva en la Figura 6-6(d):

$$Q(t+1) = (J)Q' + (K')Q$$

Las variables de entrada J y K se encierran en paréntesis, de tal manera que no se confunda los términos AND de la ecuación característica con la convención de dos letras las cuales se han usado para representar las variables de entrada de los flip-flops.

El circuito secuencial puede derivarse directamente de las ecuaciones de estado sin tener que dibujar la tabla de excitación. Esto se hace por medio de un proceso de apareamiento entre la ecuación de estado para cada flip-flop y la ecuación general característica del flip-flop JK. El proceso de apareamiento consiste en manipular cada ecuación de estado hasta que esté en la forma de ecuación característica. Una vez que se hace esto, las funciones para las entradas J y K pueden ser extractadas y simplificadas. Esto debe hacerse para cada ecuación de estado listada, y su nombre de variable de flip-flop A, B, C, etc., debe remplazar la letra Q en la ecuación característica.

*Esta parte puede omitirse sin pérdida de continuidad.

Una ecuación de estado dada para $Q(t+1)$ puede expresarse como función de Q y Q'. A menudo, o Q o Q' o ambas estarían ausentes en la expresión de Boole. Es necesario entonces manipular la expresión algebraicamente hasta que Q y Q' se incluyan en las posibilidades que pueden encontrarse.

EJEMPLO 6-5: Diseñar un circuito secuencial con los flip-flops JK para satisfacer las siguientes ecuaciones de estado:

$$A(t+1) = A'B'CD + A'B'C + ACD + AC'D'$$
$$B(t+1) = A'C + CD' + A'BC'$$
$$C(t+1) = B$$
$$D(t+1) = D'$$

Las funciones de entrada del flip-flop A se derivan por este método rearreglando la ecuación de estado y apareándola con la ecuación característica de la siguiente manera:

$$\begin{aligned} A(t+1) &= (B'CD + B'C)A' + (CD + C'D')A \\ &= (J)A' + (K')A \end{aligned}$$

De la igualdad de estas dos funciones, se deducen las funciones de entrada del flip-flop A como:

$$J = B'CD + B'C = B'C$$
$$K = (CD + C'D')' = CD' + C'D$$

La ecuación de estado para el flip-flop B puede rearreglarse de la siguiente manera:

$$B(t+1) = (A'C + CD') + (A'C')B$$

Sin embargo, esta forma no es adecuada para aparearla con la ecuación característica porque la variable B' está faltando. Si a la primera cantidad en paréntesis se le aplica la función AND conjuntamente con $(B' + B)$, la ecuación permanece igual, pero con la variable B' incluida. Entonces:

$$\begin{aligned} B(t+1) &= (A'C + CD')(B' + B) + (A'C')B \\ &= (A'C + CD')B' + (A'C + CD' + A'C')B \\ &= (J)B' + (K')B \end{aligned}$$

De la igualdad de estas dos funciones, se deducen las funciones de entrada para el flip-flop B:

$$J = A'C + CD'$$
$$K = (A'C + CD' + A'C')' = AC' + AD$$

La ecuación de estado para el flip-flop C puede manipularse como sigue:

$$C(t+1) = B = B(C' + C) = BC' + BC$$
$$= (J)C' + (K')C$$

Las funciones de entrada del flip-flop C son:

$$J = B$$
$$K = B'$$

Finalmente, la ecuación de estado del flip-flop D puede ser manipulada para el propósito de apareamiento de la siguiente manera:

$$D(t+1) = D' = 1.D' + 0.D$$
$$= (J)D' + (K')D$$

lo cual da la función de entrada:

$$J = K = 1$$

Las funciones de entrada derivadas pueden acumularse y listarse conjuntamente. La convención de dos letras para designar la variable de entrada del flip-flop, no usada en la anterior derivación, se usa a continuación:

$JA = B'C$	$KA = CD' + C'D$
$JB = A'C + CD'$	$KB = AC' + AD$
$JC = B$	$KC = B'$
$JD = 1$	$KD = 1$

El procedimiento de diseño introducido aquí es un método alterno para determinar las funciones de entrada del flip-flop del circuito secuencial cuando se usan flip-flops JK. Para usar este procedimiento cuando un diagrama de estado o tabla de estado se especifica inicialmente, es necesario que las ecuaciones de estado se deriven por el procedimiento esbozado en la Sección 6-4. El método de la ecuación de estado para encontrar las funciones de entrada del flip-flop puede extenderse para cubrir estados sin usar los cuales se consideran como funciones de no importa. Los términos mínimos de no importa se escriben en la forma de una ecuación de estado y se manipulan hasta que estén en la forma de la ecuación característica para el flip-flop particular considerado. Las funciones J y K en la ecuación de estado de no importa se toman como términos mínimos de no importa cuando se simplifican las funciones de entrada de un flip-flop particular.

REFERENCIAS

1. Marcus, M. P., *Switching Circuits for Engineers*, 3a. ed. Englewood Cliffs, N.J.: Prentice-Hall, 1975.

2. McCluskey, E. J., *Introduction to the Theory of Switching Circuits.* Nueva York: McGraw-Hill Book Co., 1965.

3. Miller, R. E., *Switching Theory,* dos volúmenes. Nueva York: John Wiley and Sons, 1965.

4. Krieger, M., *Basic Switching Circuit Theory.* Nueva York: The Macmillan Co., 1967.

5. Hill, F. J. y G. R. Peterson, *Introduction to Switching Theory and Logical Design.* Nueva York: John Wiley and Sons, 1974.

6. Givone, D. D., *Introduction to Switching Circuit Theory.* Nueva York: McGraw-Hill Book Co., 1970.

7. Kohavi, Z., *Switching and Finite Automata Theory.* Nueva York: McGraw-Hill Book Co., 1970.

8. Phister M., *The Logical Design of Digital Computers.* Nueva York: John Wiley and Sons, 1958.

9. Paull, M. C. y S. H. Unger, "Minimizing the Number of States in Incompletely Specified Sequential Switching Functions". *IRE Trans. on Electronic Computers,* Vol. EC-8, No. 3 (setiembre 1959), 356-66.

10. Hartmanis, J., "On the State Assignment Problem for Sequential Machines I." *IRE Trans. on Electronic Computers,* Vol. EC-10, No. 2 (junio 1961), 157-65.

11. McCluskey, E. J. y S. H. Unger, "A Note on the Number of Internal Assignments for Sequential Circuits". *IRE Trans. on Electronic Computer,* Vol. EC-8, No. 4 (diciembre 1959), 439-40.

PROBLEMAS

6-1. Dibuje el diagrama lógico de un flip-flop *RS* temporizado con cuatro compuerpuertas NAND.

6-2. Dibuje el diagrama lógico de un flip-flop *D* temporizado con compuertas AND y NOR.

6-3. Demuestre que el flip-flop *D* temporizado de la Figura 6-5(a) puede simplificarse en una compuerta.

6-4. Considere un flip-flop *JK'* es decir un flip-flop *JK* con un inversor entre la entrada externa *K'* y la entrada interna *K*.

(a) Obtenga la tabla característica del flip-flop.

(b) Obtenga la ecuación característica.

(c) Demuestre que atando las dos entradas externas entre sí se forma un flip-flop *D*.

6-5. Un flip-flop con entrada principal de puesta a uno, tiene entradas de puesta a uno y de puesta a cero. Esta difiere de un flip-flop *RS* convencional en que si se pone a uno y a cero simultáneamente, el flip-flop como resultado se pondrá a uno.

(a) Obtenga la tabla característica y ecuación característica de un flip-flop con dominio de puesta a uno (set-dominate).

(b) Obtenga el diagrama lógico de un flip-flop con dominio de puesta a uno asincrónico.

6-6. Obtenga el diagrama lógico de un flip-flop JK maestro esclavo con compuertas AND y NOR. Incluya una provisión para poner a uno y a cero el flip-flop asincrónicamente (sin reloj).

6-7. Este problema investiga la operación del flip-flop JK maestro esclavo a través de la transición binaria en las compuertas internas de la Figura 6-11. Evalúe los valores binarios (0 ó 1) en las salidas de las nueve compuertas cuando las entradas del circuito van a través de la siguiente secuencia:

(a) $CP = 0$, $Y = 0$, $Q = 0$ y $J = K = 1$.

(b) Después de que CP vaya a 1 (Y debe ir a uno; Q permanece en 0).

(c) Después de que CP vaya a 0 e inmediatamente después J irá a 0 (Q debe ir a 1; Y queda sin afectarse).

(d) Después de que CP vaya a 1 de nuevo (Y debe ir a 0).

(e) Después de que CP vaya de vuelta a 0 e inmediatamente después de eso K vaya a 0 (Q debe ir a 0).

(f) Todos los pulsos que se suceden no tienen efecto siempre y cuando J y K permanezcan en 0.

6-8. Dibuje el diagrama lógico (mostrando todas las compuertas de un flip-flop D maestro esclavo. Use compuertas NAND.

6-9. Conecte un terminal de puesta a cero (clear) asincrónico o las entradas de las compuertas 2 y 6 del flip-flop de la Figura 6-12.

(a) Demuestre que cuando el terminal de puesta a cero es 0, el flip-flop se pone a cero y permanece así independientemente de dos valores de las entradas CP y D.

(b) Demuestre que cuando la entrada de puesta a cero es 1, no tiene efecto en las operaciones normales temporizadas.

6-10. El sumador completo de la Figura P6-10 recibe dos entradas externas x y y; la tercera entrada z viene de la salida del flip-flop D. El arrastre de salida (carry output) se trasfiere al flip-flop en cada pulso de reloj. La salida externa S da la suma de x, y y z. Obtenga la tabla de estado y el diagrama de estado del circuito secuencial.

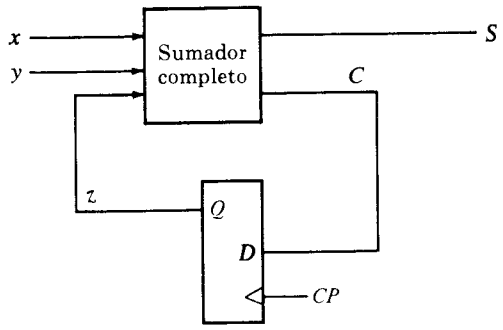


Figura P6-10

6-11. Deduzca la tabla de estado y diagrama de estado del circuito secuencial de la Figura P6-11. ¿Cuál es la función del circuito?

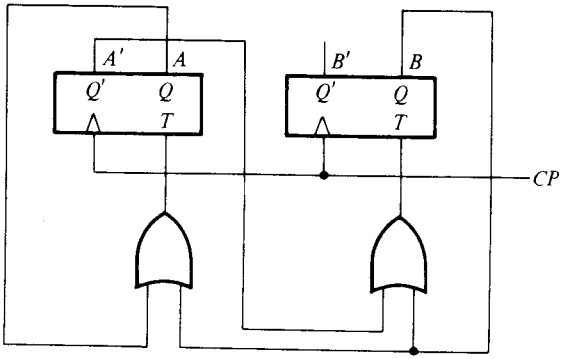


Figura P6-11

6-12. Un circuito secuencial tiene cuatro flip-flops A, B, C y D y una entrada x. Este se describe por medio de las siguientes ecuaciones:

$$A(t+1) = (CD' + C'D)x + (CD + C'D')x'$$
$$B(t+1) = A$$
$$C(t+1) = B$$
$$D(t+1) = C$$

(a) Obtenga la secuencia de estados cuando $x = 1$, comenzando desde el estado $ABCD = 0001$.

(b) Obtenga la secuencia de estados cuando $x = 0$ comenzando desde el estado $ABCD = 0000$.

6-13. Un circuito secuencial tiene dos flip-flops (A y B), dos entradas x y y, y una salida z. Las funciones de entrada del flip-flop y las funciones de salida del circuito son las siguientes:

$$JA = xB + y'B' \qquad KA = xy'B'$$
$$JB = xA' \qquad KB = xy' + A$$
$$z = xyA + x'y'B$$

Obtenga el diagrama lógico, la tabla de estado, el diagrama de estado y las ecuaciones de estado.

6-14. Reduzca el número de estados en la siguiente tabla de estado y tabule la tabla de estado reducida.

Estado presente	Estado siguiente		Salida	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

6-15. Comenzando con el estado a de la tabla de estado en el Problema 6-14, encuentre la secuencia de salida generada con la secuencia de entrada

01110010011.

6-16. Repita el Problema 6-15 usando la tabla reducida del Problema 6-14. Demuestre que se obtiene la misma secuencia de salida.

6-17. Substituya la asignación binaria 2 de la Tabla 6-5 a los estados en la Tabla 6-4 y obtenga la tabla de estado binario. Repítalo con la asignación binaria 3.

6-18. Obtenga la tabla de excitación del flip-flop JK' descrita en el Problema 6-4.

6-19. Obtenga la tabla de excitación de un flip-flop con dominio de puesta a uno (set-dominate) descrita en el Problema 6-5.

6-20. Un circuito secuencial tiene una entrada y una salida. El diagrama de estado se muestra en la Figura P6-20. Diseñe un circuito secuencial con (a) flip-flops T, (b) flip-flops RS y (c) flip-flops JK.

6-21. Diseñe el circuito de un registro de 3 bits que convierte el número acumulado en el registro a su valor de complemento de 2 cuando la entrada $x = 1$. Los flip-flops del registro son del tipo RST. Este flip-flop tiene tres entradas: dos entradas tienen características RS y una tiene características T. Las entradas RS se usan para trasferir el número de 4 bits cuando una entrada $y = 1$. Use la entrada T para la conversión.

6-22. Repita el Ejemplo 6-1 con la asignación binaria 3 de la Tabla 6-5. Use los flip-flops JK.

6-23. Diseñe un contador BDC con flip-flops JK.

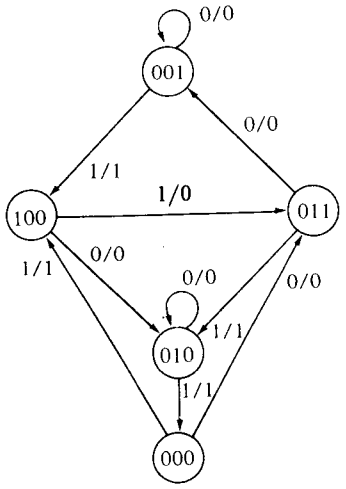


Figura P6-20

6-24. Diseñe un contador que cuente dígitos decimales de acuerdo al código 2, 4, 2, 1, (Tabla 1-2). Use flip-flops T.

6-25. Diseñe los contadores binarios que tienen la siguiente secuencia binaria repetida. Use flip-flops JK.

(a) 0, 1, 2

(b) 0, 1, 2, 3, 4

(c) 0, 1, 2, 3, 4, 5, 6

6-26. Diseñe un contador con la siguiente secuencia binaria: 0, 1, 3, 2, 6, 4, 5, 7 y repetición. Use flip-flops RS.

6-27. Diseñe un contador con la siguiente secuencia binaria: 0, 1, 3, 7, 6, 4 y repetición. Use flip-flops T.

6-28. Diseñe un contador con la siguiente secuencia binaria: 0, 4, 2, 1, 6 y repetición. Use flip-flops JK.

6-29. Repita el Ejemplo 6-5 usando flip-flops D.

6-30. Verifique el circuito obtenido en el Ejemplo 6-5 usando el método de la tabla de excitación.

6-31. Diseñe el circuito secuencial descrito por medio de las siguientes ecuaciones de estado. Use flip-flops JK.

$$A(t+1) = xAB + yA'C + xy$$
$$B(t+1) = xAC + y'BC'$$
$$C(t+1) = x'B + yAB'$$

6-32. (a) Deduzca las ecuaciones de estado para el circuito secuencial especificado en la Tabla 6-6, Sección 6-5. Liste los términos de no importa. (b) Deduzca las funciones de entrada de los flip-flops a partir de las ecuaciones de estado (y los términos de no importa) usando el método esbozado en el Ejemplo 6-5. Use flip-flops JK.

Registros, contadores y unidad de memoria

7

7-1 INTRODUCCION

Un circuito secuencial temporizado consiste en un grupo de flip-flops y compuertas combinacionales conectados para formar un camino de realimentación. Los flip-flops son esenciales porque, en su ausencia, el circuito se reduce a un circuito puramente combinacional (siempre y cuando no haya un camino de realimentación). Un circuito con flip-flops solamente se considera un circuito secuencial aun en la ausencia de compuertas combinacionales.

Un circuito MSI que tiene celdas de almacenamiento dentro de él es por definición un circuito secuencial. Los circuitos MSI que incluyen flip-flops u otras celdas de almacenamiento se clasifican comúnmente por la función que ellas realizan en vez de por el nombre "circuito secuencial". Estos circuitos MSI se clasifican en una de tres categorías: registros, contadores o memorias de acceso aleatorio. Este capítulo presenta varios registros y contadores obtenibles en la forma de CI y se explica su operación. La organización de la memoria de acceso aleatorio se presenta también.

Un *registro* es un grupo de celdas de almacenamiento binario capaz de retener información binaria. Un grupo de flip-flops constituyen un registro ya que cada flip-flop es una celda binaria que acumula un bit de información. Un registro de n-bits tiene un grupo de n flip-flops y tiene capacidad de acumular cualquier información binaria que contiene n bits. Además de los flip-flops, un registro puede tener compuertas combinacionales que ejecutan ciertas tareas de procesamiento de datos. En su definición más general, un registro consiste en un grupo de flip-flops y compuertas que afectan su transición. El flip-flop retiene información binaria y las compuertas controlan cuándo y cómo se trasfiere la nueva información al registro.

Los contadores se introdujeron en la Sección 6-8. Un contador es esencialmente un registro que pasa por una secuencia predeterminada de estados después de la aplicación de pulsos de entrada. Las compuertas en un contador se conectan de tal manera que se produce una secuencia preestablecida de estados binarios en el registro. Aunque los contado-

res son un tipo especial de registro, es común diferenciarlos dándoles un nombre especial.

Una unidad de memoria es una colección de celdas de almacenamiento conjuntamente con los circuitos asociados necesarios para trasferir la información de entrada y salida. Una memoria de acceso aleatorio (RAM) difiere de una memoria de solo lectura (ROM) en que una RAM puede trasferir la información acumulada hacia afuera (lectura) y también es capaz de recibir nueva información para almacenamiento (escritura). Un nombre más adecuado para tal memoria podría ser *memoria de lectura y escritura*.

Los registros, los contadores y las memorias se usan externamente en el diseño de sistemas digitales en general y computadores digitales en particular. Los registros pueden usarse también para facilitar el diseño de circuitos secuenciales. Los contadores son útiles para generar variables de tiempo para temporizar y controlar las operaciones en un sistema digital. Las memorias son esenciales para almacenar los programas y los datos en un computador digital. El conocimiento de las operaciones de estos componentes es indispensable para la comprensión de la organización y diseño de los sistemas digitales.

7-2 REGISTROS

Varios tipos de registros están disponibles en circuitos MSI. El circuito más simple es aquel que consiste en flip-flops sin ninguna compuerta externa. La Figura 7-1 muestra tal registro construido con cuatro flip-flops tipo D y un pulso de reloj común de entrada. El pulso de reloj de entrada, CP, habilita todos los flip-flops de manera que la información disponible al presente en las cuatro entradas pueda ser trasferida al registro de 4 bits. Las cuatro salidas pueden ser cateadas para obtener la información acumulada en el registro.

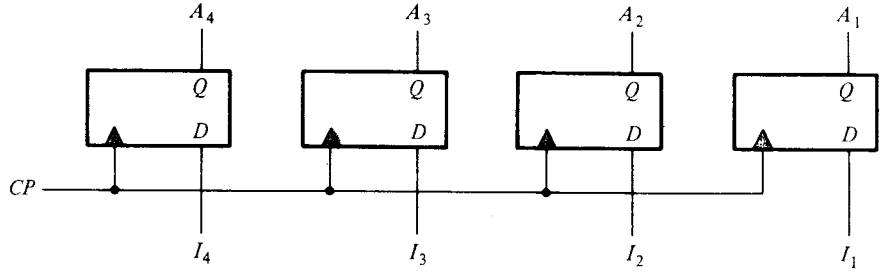


Figura 7-1 Registro de 4 bits

La forma en que los flip-flops de un registro se disparan es de suprema importancia. Si los flip-flops se construyen con compuertas retenedoras tipo D (gated D-type latches) como en la Figura 6-5, la información presente en la entrada (D) de datos se trasfiere a la salida Q cuando el habilitador (CP) es 1. Cuando CP va a cero, la información que estaba

presente en la entrada de datos justamente antes de la transición es retenida en la salida Q. En otras palabras los flip-flops son sensibles a la duración del pulso, y el registro se habilita durante el tiempo que $CP = 1$. Un registro que responde a la duración del pulso se llama comúnmente *compuerta retenedora* (gated latch), y la entrada CP se marca con la variable G (en vez de CP). Los retenedores son útiles para almacenamiento temporal de la información binaria que se va a trasferir a un destino externo. No se deben usar en el diseño de circuitos secuenciales que tienen conexiones de realimentación.

Como se explica en la Sección 6-3, un flip-flop puede ser usado en el diseño de circuitos secuenciales temporizados siempre y cuando sean sensibles a la transición del pulso en vez de la duración del pulso. Esto significa que los flip-flops en el registro deben ser del tipo de disparo por flanco o maestro esclavo. Normalmente no es posible distinguir en un diagrama lógico cuándo un flip-flop es un retenedor de compuerta, se dispara por flanco ó es maestro esclavo, porque los símbolos gráficos de las tres son iguales. La distinción debe hacerse a partir del nombre dado a la unidad. Un grupo de flip-flops sensibles a duración de pulso se llaman por lo general un *retenedor* (latch), mientras que un grupo de flip-flops sensibles a transición de pulso se llaman un *registro*.* Un registro puede ser siempre remplazado por un retenedor, si el remplazo se hace con cuidado con el fin de asegurarse que las salidas del retenedor nunca vayan a otras entradas de flip-flops que estén activadas con el mismo pulso de reloj común. En las discusiones subsiguientes, se asumirá siempre que cualquier grupo de flip-flops dibujados constituye un *registro* y que todos los flip-flops son del tipo de disparo por flanco o maestro esclavo. Si el registro es sensible a la duración del pulso, será tratado como un *retenedor* (latch).

Registro con carga en paralelo

La trasferencia de nueva información a un registro se denomina como la *carga* del registro. Si todos los bits del registro se cargan simultáneamente con un solo pulso de reloj, se dice que la carga se hace en paralelo. Un pulso aplicado a la entrada CP del registro de la Figura 7-1 cargará todas las cuatro entradas en paralelo. En esta configuración, el pulso de reloj debe aislarse del terminal CP si el contenido del registro se debe dejar sin cambio. En otras palabras, la entrada CP actúa como una señal de habilitación la cual controla la carga de la nueva información al registro. Cuando CP va a 1, la información de entrada se carga al registro. Si CP permanece en 0, el contenido del registro no cambia. Nótese que el cambio de estado en la entrada ocurre en el flanco positivo del pulso. Si el flip-flop cambia de estado en el flanco negativo, habrá un pequeño círculo debajo del símbolo de triángulo en la entrada CP del flip-flop.

La mayoría de los sistemas digitales tienen un generador de pulsos de reloj maestro que suministra un tren de pulsos de reloj. Todos los pulsos de reloj se aplican a todos los flip-flops y registros en el sistema. El

*Por ejemplo el CI tipo 7475 es un retenedor de 4 bits, mientras que el CI tipo 74175 es un registro de 4 bits.

generador de pulsos de reloj maestro actúa como una bomba que suministra un ritmo a todas las partes del sistema. Una señal de control separada decide entonces qué pulso de reloj específico tendrá un efecto en un registro particular. En tal sistema, los pulsos de reloj deben ser, conjuntamente con la señal de control, aplicados a una compuerta AND para que la salida de esta última se aplique al terminal *CP* del registro mostrado en la Figura 7-1. Cuando la señal de control es 0, la salida de la compuerta AND será 0 y la información almacenada en el registro permanecerá sin cambiar. Solamente cuando la señal de control es un 1, el pulso de reloj pasará por la compuerta AND y llegará al terminal *CP* para que la nueva información se cargue al registro. Tal variable de control se llama *terminal de control de carga*.

El colocar una compuerta AND en el camino de los pulsos de reloj significa que la lógica se ejecuta con pulsos de reloj. El agregar compuertas lógicas produce retardos de propagación entre el generador del pulso maestro y las entradas de reloj de los flip-flops. Para sincronizar comple-

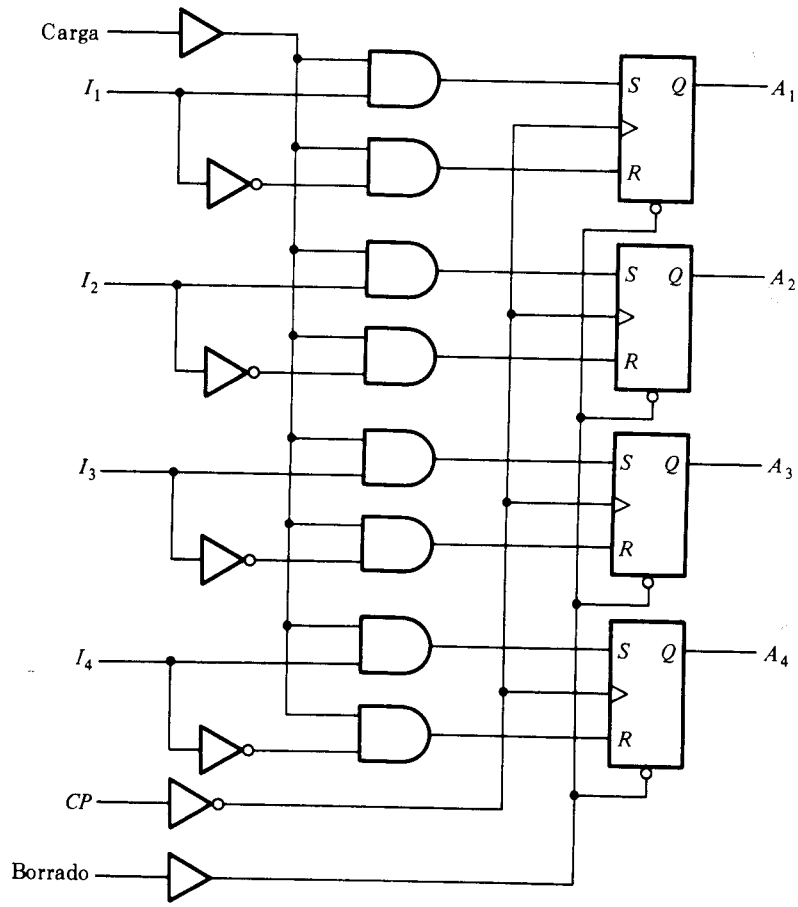


Figura 7-2 Registro de 4 bits con carga en paralelo

tamente un sistema es necesario asegurarse que todos los pulsos de reloj llegan al mismo tiempo a todas las entradas de todos los flip-flops de tal manera que todas cambian simultáneamente. Al ejecutar lógica con pulsos de reloj se introducen demoras variables que pueden sacar al sistema de sincronismo. Por esta razón, es aconsejable (pero no necesario siempre y cuando la demora no se tenga en cuenta) aplicar pulsos de reloj directamente a todos los flip-flops y controlar la operación del registro con otras entradas, tales como las entradas S y R de un flip-flop RS.

Un registro de 4 bits con un terminal de control de carga a base de flip-flops RS se muestra en la Figura 7-2. El terminal CP del registro recibe pulsos sincronizados continuos los cuales se aplican a todos los flip-flops. El inversor en el camino de CP causa que todos los flip-flops se dispa-

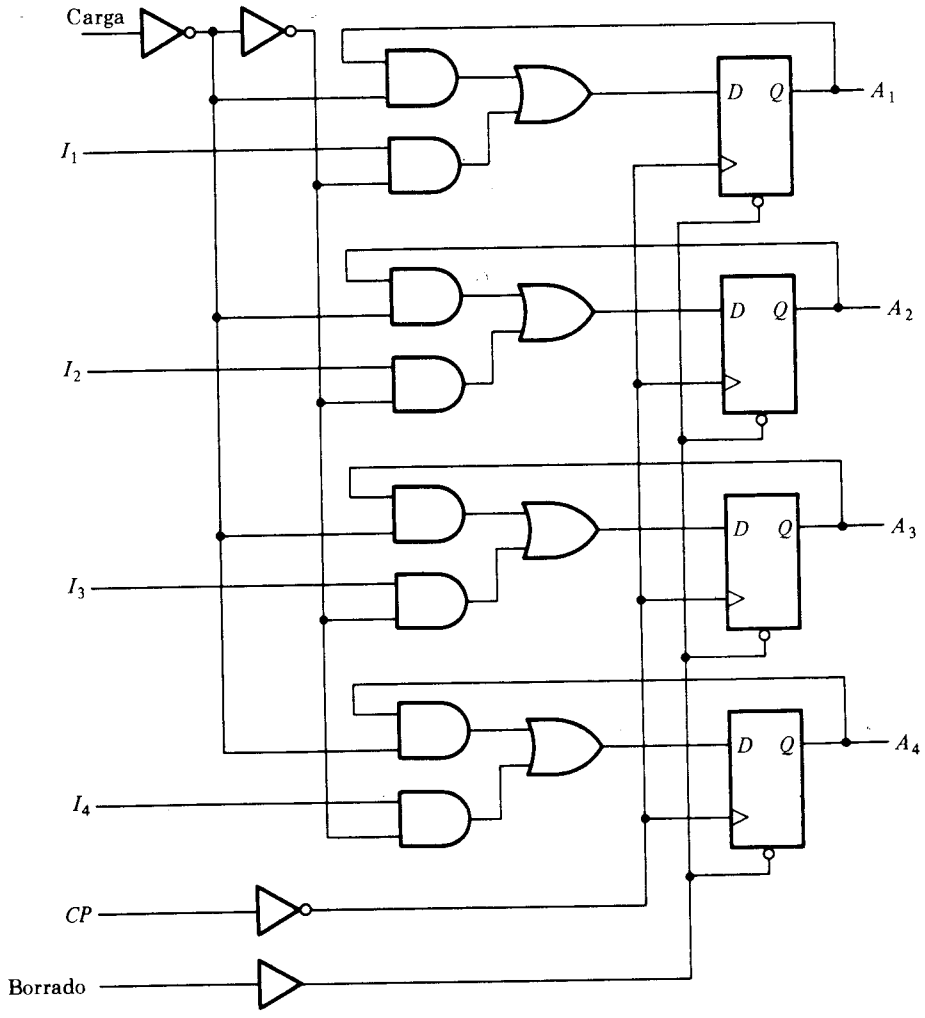


Figura 7-3 Registro con carga en paralelo con flip-flops D

ren por el flanco negativo de los pulsos entrantes. El propósito del inversor es reducir la carga del generador de pulsos maestros. Esto es debido a que el terminal *CP* se conecta solamente a una compuerta (el inversor) en vez de a las entradas de las cuatro compuertas que se hubieran podido necesitar si las conexiones se hubieran hecho directamente a los terminales de reloj de los flip-flops (marcados con pequeños triángulos).

El terminal de *borrado* (clear) o de puesta a cero va a un terminal especial en cada flip-flop a través de una compuerta separadora no inversora (noninverting buffer gate). Cuando este terminal va a 0 el flip-flop se borra asincrónicamente. La entrada de puesta a cero se usa para llevar al registro a ceros antes de la operación en cadencia. La entreda de puesta a cero debe mantenerse en 1 durante las operaciones normales temporizadas (ver Figura 6-14).

El terminal de *carga* pasa a través de una compuerta separadora (para reducir la carga) y a través de una serie de compuertas AND va a los terminales R y S de cada flip-flop. Aunque los pulsos de reloj están presentes continuamente, en el terminal de carga que controla la operación del registro. Las dos compuertas AND y el inversor asociado con cada terminal I determinan los valores de R y S. Si el terminal de carga es 0, ambos R y S son cero, y no ocurrirá cambio de estado con ningún pulso de reloj. Así, la señal del terminal de carga es una variable de control la cual puede prevenir cualquier cambio de información en el registro siempre que esté su señal en 0. Cuando el control de carga vaya a 1, las entradas I_1 hasta I_4 especificarán qué información binaria se carga al registro en el siguiente pulso de reloj. Para cada I que sea igual a 1, las entradas del flip-flop correspondientes son $S = 1$, $R = 0$. Para cada I que sea igual a 0, las entradas de los flip-flops correspondientes son $S = 0$, $R = 1$. Así, el valor de la entrada se trasfiere al registro, si el terminal de carga es 1, el terminal de borrado es 1, y el pulso de reloj pasa de 1 a 0. Este tipo de trasferencia se llama trasferencia de *carga en paralelo* porque todos los bits se cargan simultáneamente. Si la compuerta separadora asociada con la entrada de carga se cambia a una compuerta inversor, entonces el registro se carga cuando el terminal de carga es 0 y se inhibe cuando es 1.

Un registro con carga paralela puede ser construido con flip-flops D como se muestra en la Figura 7-3. Los terminales de reloj y de borrado son los mismos que antes. Cuando el terminal de carga es 1, las entradas I se trasfieren al registro en el pulso siguiente de reloj. Cuando el terminal de carga es 0, las entradas del circuito se inhiben y los flip-flops D se cargan con su valor presente, manteniendo así el contenido del registro. La conexión de realimentación en cada flip-flop es necesaria cuando se usa del tipo D ya que el flip-flops tipo D no tiene una condición de entrada de "no cambio". La entrada D determina el siguiente estado de la salida con cada pulso de reloj. Para dejar la salida sin cambiar, es necesario hacer la entrada D igual a la salida presente Q en cada flip-flop.

Configuración con lógica secuencial

Se trató en el Capítulo 6 que un circuito secuencial temporizado consiste en un grupo de flip-flops y compuertas combinacionales. Como los registros

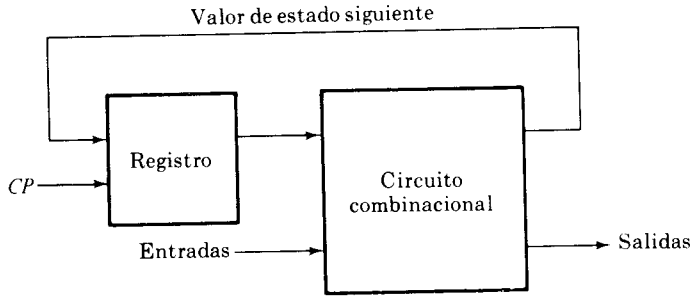


Figura 7-4 Diagrama de bloque de un circuito secuencial

están disponibles fácilmente como circuitos MSI, es conveniente algunas veces emplear un registro como parte de un circuito secuencial. Un diagrama de bloque de un circuito secuencial que usa un registro se muestra en la Figura 7-4. El estado presente del registro y las entradas externas determinan el siguiente estado del registro y los valores de las salidas externas. Parte del circuito combinacional determina el siguiente estado y la otra parte genera las salidas. El siguiente valor del estado del circuito combinacional se carga en el registro con un pulso de reloj. Si el registro tiene un terminal de carga, se debe establecer a 1; de otra manera, si el registro no tiene terminal de carga (como en la Figura 7-4), el siguiente valor del estado será trasferido automáticamente en cada pulso de reloj.

La parte de circuito combinacional de un circuito secuencial puede ser ejecutada por cualquiera de los métodos discutidos en el Capítulo 5. Se puede construir con compuertas SSI con ROM, o con un arreglo lógico programable (PLA). Usando un registro, es posible reducir el diseño de un cicuito secuencial al de un circuito combinacional conectado a un registro.

EJEMPLO 7-1: Diseñar un circuito secuencial cuya tabla de estado se lista en la Figura 7-5(a).

La tabla de estado especifica dos flip-flops A_1 y A_2 una entrada x y una entrada y. El siguiente estado e información de salida se obtiene directamente de la tabla:

$$A_1(t+1) = \Sigma(4, 6)$$
$$A_2(t+1) = \Sigma(1, 2, 5, 6)$$
$$y(A_1, A_2, x) = \Sigma(3, 7)$$

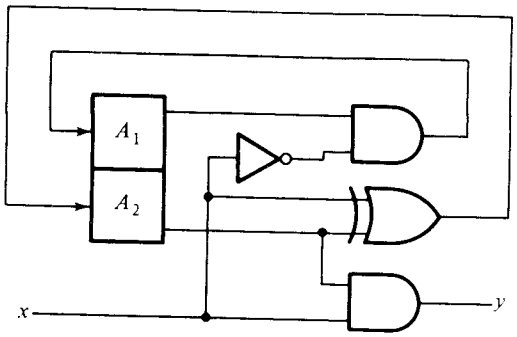
Los valores de términos mínimos son para las salidas A_1, A_2 y x, los cuales son el estado presente y las variables de entrada. Las funciones para el estado siguiente y la salida pueden ser simplificadas por medio de mapas para dar:

$$A_1(t+1) = A_1x'$$
$$A_2(t+1) = A_2 \oplus x$$
$$y = A_2x$$

El diagrama lógico se muestra en la Figura 7-5(b).

Estado presente		Entrada	Estado siguiente		Salida
A_1	A_2	x	A_1	A_2	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) Tabla de estado



(b) Diagrama lógico

Figura 7-5 Ejemplo de una configuración de un circuito secuencial

Tabla de verdad de la ROM

Dirección			Salidas		
1	2	3	1	2	3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

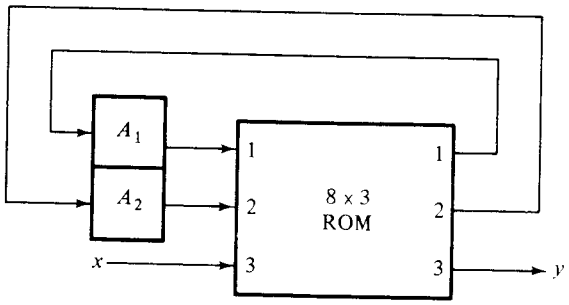


Figura 7-6 Circuito secuencial que usa un registro y una ROM

EJEMPLO 7-2: Repítase el Ejemplo 7-1 pero úsese ahora una ROM y un registro.

La ROM puede usarse para configurar el circuito combinacional y el registro suministrará los flip-flops. El número de entradas de la ROM es igual al número de flip-flops más el número de entradas externas. El número de salidas de la ROM es igual al número de flip-flops más el número de salidas externas. En este caso se tienen tres entradas y tres salidas de la ROM; de tal forma que su tamaño puede ser de 8×3. La configuración se muestra en la Figura 7-6. La tabla de verdad de la ROM es idéntica a la tabla de estado con "estado presente" y "entradas" especificando la dirección de la ROM y el "estado siguiente" y las "salidas" que especifican las salidas de la ROM. Los valores del estado siguiente deben ser conectados de las salidas de la ROM a las entradas del registro.

7-3 REGISTROS DE DESPLAZAMIENTO

Un registro capaz de desplazar su información binaria hacia la izquierda o hacia la derecha se llama *registro de desplazamiento*. La configuración

lógica de un registro de desplazamiento consiste en una cadena de flip-flops conectados en cascada, con la salida de un flip-flop conectado a la entrada del siguiente. Todos los flip-flops reciben un pulso de reloj común el cual causa el deplazamiento de un estado al siguiente.

El registro de desplazamiento más sencillo es aquel que usa solamente flip-flops como se muestra en la Figura 7-7. La salida Q de un flip-flop dado, se conecta a la entrada D del flip-flop a la derecha. Cada pulso de reloj desplaza el contenido del registro un bit en posición a la derecha. La *entrada serial* determina qué va en el flip-flop de la extrema izquierda durante el desplazamiento. La *salida serial* se toma de la salida del flip-flop de la extrema derecha después de la aplicación de un pulso. Aunque este registro desplace su contenido a la derecha, si se voltea la página se observa que el registro desplaza su contenido a la izquierda. Así un registro de desplazamiento unidireccional puede funcionar como un registro de desplazamiento a la derecha o a la izquierda.

El registro en la Figura 7-7 desplaza un contenido con cada pulso de reloj durante el flanco negativo del pulso de transición. (Esto es indicado por el pequeño círculo asociado con la entrada de reloj en todos los flip-flops.) Si se requiere controlar el desplazamiento de tal manera que ocurra solamente con ciertos pulsos pero no con otros, se debe controlar el terminal CP del registro. Se mostrará más adelante, que las operaciones de desplazamiento pueden ser controladas a través de las entradas D de los flip-flops en vez de a través del terminal CP. Si se usa el registro de la Figura 7-7 se puede controlar el desplazamiento por medio de una compuerta AND como se muestra a continuación.

Trasferencia en serie

Se dice que un sistema digital opera en modo serie cuando la información se trasfiere y se manipula un bit en cada tiempo. El contenido de un registro se trasfiere a otro desplazando los bits de un registro al siguiente. La información se trasfiere bit a bit, uno cada vez desplazando los bits del registro fuente hacia el registro de destino.

La trasferencia en serie de la información del registro A al registro B se hace con registros de desplazamiento, como se muestra en el diagrama de bloque de la Figura 7-8(a). La salida serial (SO) del registro A va a la entrada serial (SI) del registro B. Para prevenir la pérdida de información almacenada en el registro fuente, al registro A se le hace circular su información conectando la salida serial a su terminal de entrada serial. El con-

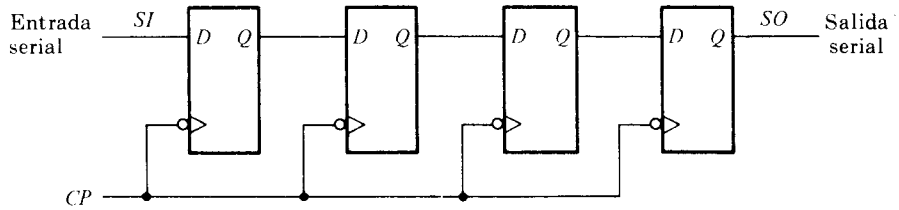
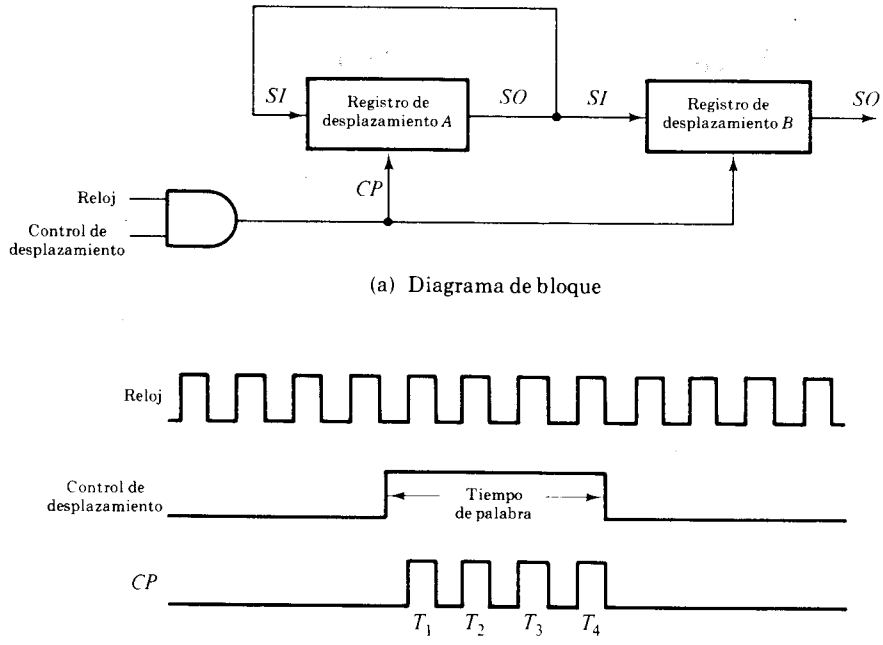


Figura 7-7 Registro de desplazamiento

tenido inicial del registro B es desplazado hacia afuera a través de su salida serial y se pierde a no ser que se desplace a un tercer registro de desplazamiento. La entrada de control de desplazamiento determina cuándo y cuántas veces se desplazan los registros. Esto se hace por medio de la compuerta AND que permite pasar los pulsos de reloj a los terminales CP solamente cuando el control de desplazamiento es 1.

Supóngase que los registros de desplazamiento tienen cuatro bits cada uno. La unidad de control que supervisa la trasferencia debe ser designada de tal forma que habilita los registros de desplazamiento por medio de la señal de control, para una duración de tiempo fija igual a cuatro pulsos de reloj. Esto se muestra en el diagrama de tiempo de la Figura 7-8(b). La señal de control de desplazamiento se sincroniza con el reloj y cambia su valor justamente después del flanco negativo del pulso de reloj. Los siguientes cuatro pulsos de reloj encuentran la señal de control de desplazamiento en el estado 1, de tal manera que la salida de la compuerta AND conectada a los terminales CP, producen los cuatro pulsos T_1, T_2, T_3 y T_4. El cuarto pulso cambia el control de desplazamiento a 0 y los registros de desplazamiento se inhabilitan.

Asúmase que el contenido binario de A antes del desplazamiento es 1011 y que el de B es 0010. La trasferencia en serie de A a B ocurrirá en



(a) Diagrama de bloque

(b) Diagrama de tiempo

Figura 7-8 Trasferencia en serie del registro A al registro B

muestra en la Figura 7-10. El bit de arrastre del sumador completo se trasfiere al flip-flop D. La salida de este flip-flop se usa entonces como arrastre de entrada para el siguiente par de bits significativos. El contenido de los dos registros de desplazamiento se desplaza a la derecha por un período de un tiempo palabra. Los bits de suma de la salida S del sumador completo pueden ser trasferidos a un tercer registro de desplazamiento. Desplazando la suma a A mientras que los bits de A se desplazan hacia el exterior, es posible usar un registro para almacenar el sumando y los bits de suma. La entrada serial (SI) del registro B es capaz de recibir un número binario nuevo mientras que los bits de suma se desplazan hacia afuera durante la suma.

La operación del sumador en serie es como sigue. Inicialmente, los registros A almacenan el sumando, el registro B almacena el otro sumando y el flip-flop de borrado se lleva a 0. Las salidas seriales (SO) de A y B suministran un par de bits significativos para el sumador completo en x y y. La salida Q de los flip-flops da el arrastre de entrada z. El control de desplazamiento a la derecha habilita ambos registros y el flip-flop del bit de arrastre; de esta manera, en el siguiente pulso de reloj ambos registros se desplazan a la derecha, el bit suma de S entra en el flip-flop de la extrema izquierda de A, y el arrastre de salida se trasfiere al flip-flop Q. El control de desplazamiento a la derecha habilita los registros por un número de pulsos de reloj iguales al número de bits en los registros. Para cada pulso de reloj sucesivo, se trasfiere un bit suma nuevo a A, un nuevo bit de arrastre a Q y ambos registros se desplazan una vez a la derecha. Este proceso continúa hasta que el control de desplazamiento a la derecha se

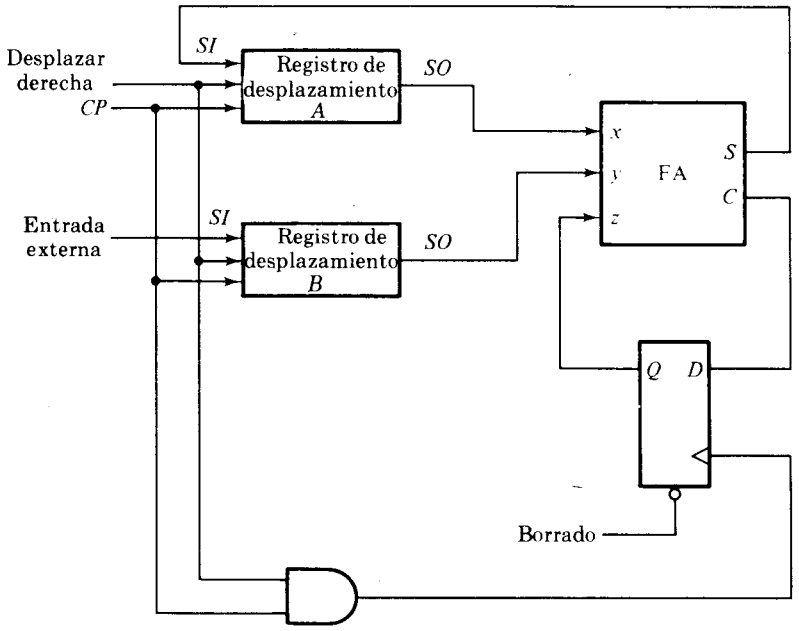


Figura 7-10 Sumador en serie

inhabilita. Así, se lleva a cabo la suma pasando cada par de bits conjuntamente con el arrastre previo a través de un circuito sumador completo sencillo y trasfiriendo la suma, un bit a la vez, al registro A.

Si el número nuevo tiene que agregarse al contenido del registro A, este número debe ser trasferido primero en serie al registro B. Repitiendo el proceso una vez más se agregará el segundo número al número previo en A.

Comparando el sumador en serie con el sumador en paralelo descrito en la Sección 5-2, se notan las siguientes diferencias. El sumador en paralelo debe usar registros con capacidad de carga en paralelo, mientras que el sumador serial usa registros de desplazamiento. El número de circuitos del sumador completo en el sumador en paralelo es igual al número de bits en los números binarios, mientras que el sumador en serie requiere solamente un circuito sumador completo y un flip-flop para el arrastre. Excluyendo los registros, el sumador en paralelo es un circuito combinacional, mientras que el sumador en serie es un circuito secuencial. El circuito secuencial en el sumador serial consiste en un circuito sumador completo y un flip-flop que acumula el arrastre de salida. Esta es una operación en serie típica porque el resultado de una operación de un tiempo de bit puede depender no solamente en las entradas presentes sino en las entradas previas.

Para mostrar que las operaciones de un tiempo del bit en los computadores en serie requieren un circuito secuencial, se diseñará el sumador serial considerando el circuito secuencial.

EJEMPLO 7-3: Diseñar un sumador en serie usando el procedimiento de lógica secuencial.

Primero se debe estipular que dos registros de desplazamiento están disponibles para almacenar los números binarios que se agregan serialmente. Las salidas seriales de los registros se designan con las variables x y y. El circuito secuencial que se va a diseñar no incluirá registros de desplazamiento, se colocarán más tarde para mostrar la unidad completa. El circuito secuencial adecuado tiene dos entradas, x y y que suministran un par de bits significativos, una salida S que genera los bits suma y el flip-flop Q para almacenar el arrastre. El estado presente de Q suministra el valor presente del arrastre. El pulso de reloj que desplaza el registro habilita el flip-flop Q para cargar el arrastre nuevo. Este arrastre es usado con el siguiente par de bits en x y y. La tabla de estado que especifica el circuito secuencial se da en la Tabla 7-3.

El estado presente de Q es el valor presente del arrastre (carry). El arrastre presente en Q se agrega conjuntamente con las entradas x y y para producir el bit suma en la salida S. El siguiente estado de Q es equivalente al arrastre de salida. Nótese que las entradas de la tabla de estado son idénticas a las entradas en la tabla de verdad del sumador completo excepto que el arrastre de entrada (input carry) está ahora presente en el estado Q

Tabla 7-3 Tabla de excitación para un sumador en serie

Estado presente	Entradas		Estado siguiente	Salida	Flip-flops de entrada	
Q	x	y	Q	S	JQ	KQ
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	0	1	0	X	0
1	1	1	1	1	X	0

y el arrastre de salida (output carry) está ahora en el estado siguiente de Q.

Si se usa un flip-flop D para Q, se obtiene el mismo circuito que el de la Figura 7-10 debido a que los requerimientos de la entrada D son los mismos que los valores del siguiente estado. Si se usa un flip-flop JK para Q, se obtienen los requerimientos de excitación de entrada listados en la Tabla 7-3. Las tres funciones de Boole de interés, son las funciones de entrada del flip-flop para JQ y KQ y la salida S. Estas funciones se especifican en la tabla de excitación y pueden ser simplificadas por medio de los mapas:

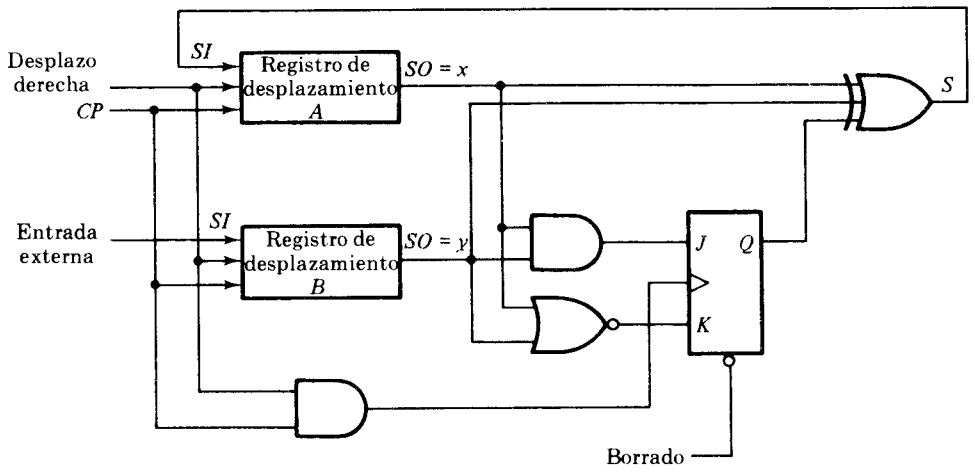


Figura 7-11 Segunda forma de un sumador en serie

$$JQ = xy$$
$$KQ = x'y' = (x + y)'$$
$$S = x \oplus y \oplus Q$$

Como se muestra en la Figura 7-11, el circuito consiste en tres compuertas y un flip-flop JK. Los dos registros de desplazamiento se incluyen también en el diagrama para mostrar el sumador completo en serie. Nótese que la salida S es una función no solamente de x y y sino también del estado presente de Q. El siguiente estado de Q es una función de valores presentes de x y y que resultan de las salidas en serie de los registros de desplazamiento.

7-4 CONTADORES DE RIZADO

Los contadores MSI vienen en dos categorías: contadores de rizado y contadores sincrónicos. En un contador de rizado, la transición de salida del flip-flop sirve como fuente para disparar los otros flip-flops. En otras palabras las salidas CP de todos los flip-flops (con excepción de la primera) se disparan no por los pulsos de entrada sino por la transición que ocurre en los otros flip-flops. En un contador sincrónico, los pulsos de entrada se aplican a todas las entradas CP de todos los flip-flops. El cambio de estado de un flip-flop en particular es dependiente del estado presente de otros flip-flops. Los contadores MSI sincrónicos se discuten en la siguiente sección. Aquí se presentan algunos contadores comunes de rizado MSI y se explica su operación.

Contador binario de rizado

Un contador binario de rizado consiste en una conexión en serie de flip-flops complementarios (tipo T ó JK), con la salida de cada flip-flop conectado a la entrada CP del siguiente flip-flop de mayor orden. El flip-flop que almacena el bit menos significativo recibe los pulsos de cuenta de entrada. El diagrama de un contador de rizado binario de 4 bits se muestra en la Figura 7-12. Todas las entradas J y K son iguales a 1. El pequeño círculo en la entrada CP indica que el flip-flop se complementa durante la transición del flanco negativo o cuando la salida a la cual está conectada

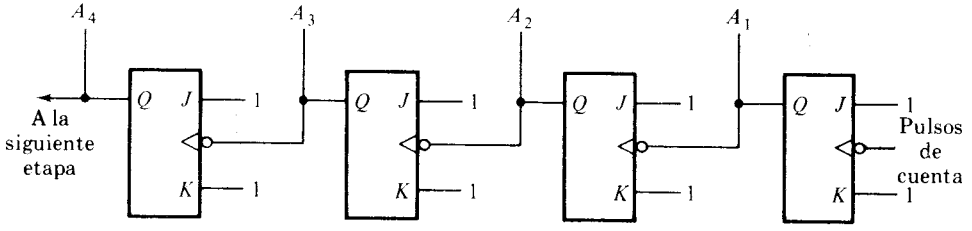


Figura 7-12 Contador binario de rizado de 4 bits

Tabla 7-4 Secuencia de cuenta para un contador binario de rizado

Secuencia de cuenta				Condiciones para complementar los flip-flops	
A_4	A_3	A_2	A_1		
0	0	0	0	Complementar A_1	
0	0	0	1	Complementar A_1	A_1 irá de 1 a 0 y complementa A_2
0	0	1	0	Complementar A_1	
0	0	1	1	Complementar A_1	A_1 irá de 1 a 0 y complementa A_2; A_2 irá de 1 a 0 y complementa A_3
0	1	0	0	Complementar A_1	
0	1	0	1	Complementar A_1	A_1 irá de 1 a 0 y complementa A_2
0	1	1	0	Complementar A_1	
0	1	1	1	Complementar A_1	A_1 irá de 1 a 0 y complementa A_2; A_2 irá de 1 a 0 y complementa A_3; A_3 irá de 1 a 0 y complementa A_4
1	0	0	0	y así sucesivamente . . .	

va de 1 a 0. Para entender la operación de un contador binario, se debe hacer referencia a la secuencia de cuenta dada en la Tabla 7-4. Es obvio que el bit de más bajo orden A_1 debe ser complementado con cada pulso de cuenta. Cada vez que A_1 va de 1 a 0, este complementa A_2. Cada vez que A_2 va de 1 a 0, este complementa A_3 y así sucesivamente. Por ejemplo, tómese la transición desde la cuenta 0111 hasta 1000. Las flechas en la tabla enfatizan las transiciones en este caso. A_1 se complementa con el pulso de cuenta. Como A_1 va de 1 a 0, este dispara A_2 y lo complementa. Como resultado, A_2 va de 1 a 0, lo cual a su turno complementa A_3. A_3 va de 1 a 0, lo cual complementa A_4. La transición de salida de A_4, si se conecta al siguiente estado, no dispara el siguiente flip-flop ya que ésta va desde 0 hasta 1. Los flip-flops cambian cada uno a su tiempo en rápida cadencia y la señal se propaga por el contador a manera de *rizo*. Los contadores de rizo se llaman algunas veces *contadores asincrónicos*.

Un contador binario con una cuenta invertida se llama un *contador binario decreciente*. En este contador la cuenta binaria se disminuye en 1 con cada pulso de cuenta de entrada. La cuenta de un contador decreciente de 4 bits comienza con el binario 15 y continúa con las cuentas binarias 14, 13, 12, . . ., 0 para pasar de nuevo a 15. El circuito de la Figura 7-12 funcionará como un contador binario decreciente si las salidas se toman de los terminales complementados Q' de todos los flip-flops. Si sólo están disponibles las salidas normales de los flip-flops, el circuito debe ser modificado ligeramente de la forma descrita a continuación.

Una lista de una secuencia de cuenta de un contador binario decreciente muestra que el bit de menor orden debe ser complementado con cada pulso de cuenta. Cualquier otro bit en la secuencia es complementado, si el bit previo de menor orden va de 0 a 1. Por tanto, el diagrama de un contador binario decreciente se ve de la misma forma que el de la Figu-

ra 7-12, teniendo en cuenta que todos los flip-flops se disparan con el flanco positivo del pulso. (El pequeño círculo en la entrada CP debe estar ausente.) Si se usan flip-flops de disparo por flanco negativo, entonces la entrada CP de cada flip-flop debe estar conectada a la salida Q' del flip-flop anterior. Entonces cuando Q vaya de 0 a 1, Q' irá de 1 a 0 y se complementará el siguiente flip-flop como se requiere.

Contador BDC de rizado

Un contador decimal sigue una secuencia de diez estados y regresa a 0 después de la cuenta de 9. Tal contador debe tener por lo menos cuatro flip-flops para representar cada dígito decimal, como un dígito decimal se representa por medio de un código binario con cuatro bits al menos. La secuencia de estados en un contador decimal se deduce del código binario usado para representar un dígito decimal. Si se usa BDC, la secuencia de estados es como se muestra en el diagrama de estado de la Figura 7-13. Esto es similar a un contador binario, excepto que el estado después de 1001 (código para el dígito decimal 9) es 0000 (código para el dígito decimal 0).

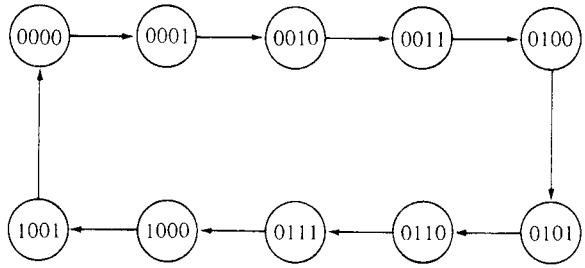


Figura 7-13 Diagrama de estado de un contador BDC decimal

El diseño para un contador de rizado decimal o para cualquier contador de rizado que no siga la secuencia binaria no es un procedimiento directo. Las herramientas formales del diseño lógico pueden servir solamente como una guía. Un producto satisfactoriamente acabado requiere la ingenuidad e imaginación del diseñador.

El diagrama lógico de un contador de rizado BDC se muestra en la Figura 7-14.* Las cuatro salidas se designan por el símbolo Q con un suscrito numérico igual a la carga binaria del bit correspondiente en el código BDC. Los flip-flops se disparan en el flanco negativo, es decir, cuando la señal CP va de 1 a 0. Nótese que la salida de Q' es aplicada a las entradas CP de ambas Q_2 y Q_8 y la salida de Q_2 se aplica a la entrada CP de Q_4. Las entradas J y K se conectan a una señal permanente de 1 a las salidas de los flip-flops como se muestra en el diagrama.

Un contador de rizado es un circuito secuencial asincrónico y no puede ser descrito por ecuaciones de Boole desarrolladas para describir circuitos secuenciales temporizados. Las señales que afectan la transición

*Este circuito es similar al CI tipo 7490.

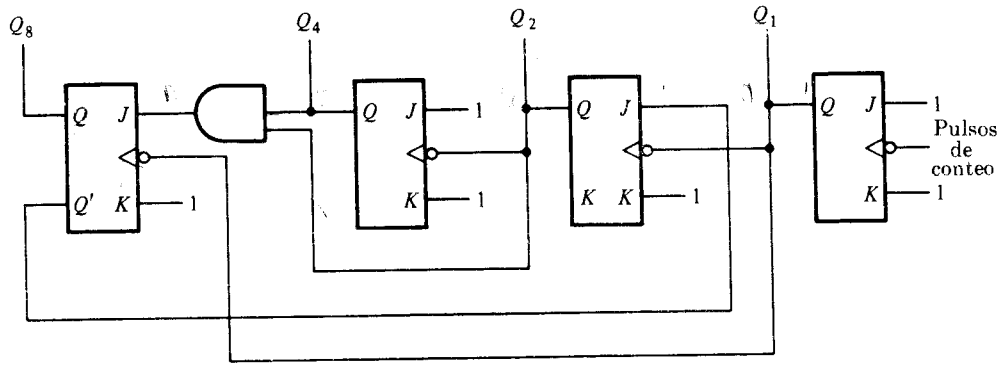


Figura 7-14 Diagrama lógico de un contador de rizado BCD

del flip-flop dependen del orden en el cual cambian de 1 a 0. La operación del contador puede ser explicada por una lista de condiciones para las transiciones de los flip-flops. Estas condiciones se deducen del diagrama lógico y del conocimiento de cómo opera un flip-flop JK. Téngase en cuenta cuando la entrada CP va de 1 a 0, el flip-flop se pone a uno si $J = 1$ y se pone a cero si $K = 1$, se complementa si $J = K = 1$, y se deja sin cambio si $J = K = 0$. Las siguientes son las condiciones para la transición de estado de cada flip-flop:

1. Q_1 se complementa en el flanco negativo de cada pulso de cuenta.
2. Q_2 se complementa si $Q_8 = 0$ y Q_1 va de 1 a 0. Q_2 se borra si $Q_8 = 1$ y Q_1 va de 1 a 0.
3. Q_4 se complementa cuando Q_2 va de 1 a 0.
4. Q_8 se complementa cuando $Q_4 Q_2 = 11$ y Q_1 va de 1 a 0. Q_8 se borra si Q_4 ó Q_2 es 0 y Q_1 va de 1 a 0.

Para verificar que estas condiciones resultan en la secuencia requerida por un contador de rizado BDC, es necesario verificar que las transi-

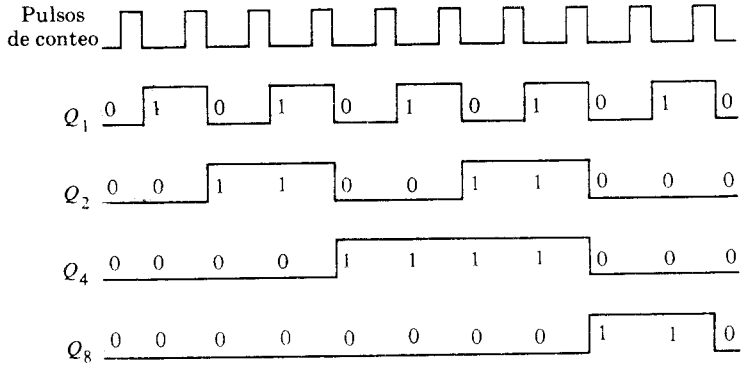


Figura 7-15 Diagrama de tiempo para el contador decimal de la Figura 7-14

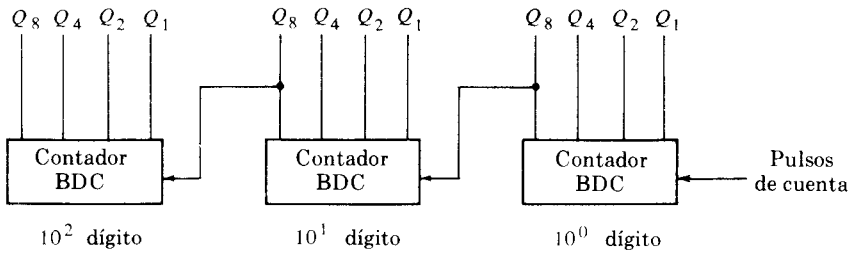


Figura 7-16 Diagrama de bloque de un contador BDC decimal de 3 décadas

ciones del flip-flop sigan ciertamente una secuencia de estados como se especifica por el diagrama de estado de la Figura 7-13. Otra manera de verificar la operación del contador es deducir el diagrama de tiempo para cada flip-flop de las condiciones listadas anteriormente. Este diagrama se muestra en la Figura 7-15, con los estados binarios listados después de cada pulso de reloj. Q_1 cambia de estado después de cada pulso de reloj. Q_2 se complementa cada vez que Q_1 va de 1 a 0 durante el tiempo en que $Q_8 = 0$. Cuando Q_8 se vuelve 1, Q_2 permanece en 0. Q_4 se complementa cada vez que Q_2 va de 1 a 0. Q_8 permanece en puesta a cero durante el tiempo en que Q_2 ó Q_4 es 0. Cuando ambas Q_2 y Q_4 se convierten en 1, Q_8 se complementa cuando Q_1 vaya de 1 a 0. Q_8 se pone a cero en la siguiente transición de Q_1.

El contador BDC de la Figura 7-14 es un contador en *década*, ya que cuenta desde 0 hasta 9. Para contar en decimal de 0 hasta 99 se necesitan dos contadores en década. Para contar desde 0 hasta 999 se necesitan tres contadores en década. Los contadores multidécada pueden construirse conectando los contadores BDC en cascada, uno para cada década. Un contador de tres décadas se muestra en la Figura 7-16. Las entradas de la segunda y tercera décadas vienen de Q_8 de la década previa. Cuando Q_8 en una década vaya de 1 a 0, esta dispara la cuenta para la década contigua de mayor orden mientras que su propia década va de 9 a 0. Por ejemplo, la cuenta siguiente a 399 será 400.

7-5 CONTADORES SINCRONICOS

Los contadores sincrónicos se distinguen de los contadores de rizado en que los pulsos de reloj se aplican a las entradas o terminales *CP* de *todos* los flip-flops. El pulso común dispara todos los flip-flops simultáneamente en vez de una a la vez en cadencia como en un contador de rizado. La decisión de cuándo se debe o no complementar un flip-flop se determina de los valores de las entradas J y K en el momento del pulso. Si $J = K = 0$, el flip-flop permanece sin cambio. Si $J = K = 1$ el flip-flop se complementa.

Un procedimiento de diseño para cualquier tipo de contador sincrónico fue presentado en la Sección 6-8. El diseño de un contador binario de 3 bits se llevó a cabo en detalle y se ilustra en la Figura 6-30. En esta sección se presentan algunos contadores típicos MSI sincrónicos y se explica su operación. Se debe tener en cuenta que no hay necesidad de diseñar un contador si se puede encontrar en la forma de CI comercial.

Contador binario

El diseño de contadores binarios sincrónicos es tan simple que no es necesario pasar por un proceso de diseño lógico secuencial riguroso. En un contador binario sincrónico, se complementa el flip-flop en la posición de menor orden con cada pulso. Esto significa que las entradas J y K deben mantenerse en la lógica 1. Un flip-flop en cualquier otra posición se complementa con un pulso siempre y cuando todos los bits en las posiciones de menor orden sean iguales a 1, porque los bits de menor orden (cuando están dados en 1) cambiarán a 0 en el siguiente pulso de cuenta. La cuenta binaria dice cuando el siguiente bit de mayor orden debe ser complementado. Por ejemplo, si el estado presente de un contador de 4 bits es $A_4A_3A_2A_1 = 0011$, la siguiente cuenta será 0100. A_1 se complementa siempre. A_2 se complementa porque el estado presente de $A_1 = 1$. A_3 se complementa porque el estado presente de $A_2A_1 = 11$. Pero A_4 no se complementa por el estado presente de $A_3A_2A_1 = 011$, lo cual no dará una condición de solo unos.

Los contadores binarios sincrónicos tienen un patrón regular y pueden fácilmente ser construidos con flip-flops complementados y compuertas. El patrón regular puede verse claramente del contador de 4 bits ilustrado en la Figura 7-17. Los terminales CP de todos los flip-flops están conectados a una fuente de pulsos de reloj común. La primera etapa A_1 tiene J y K igual a 1 si el contador está habilitado. Las otras entradas J y K son iguales a 1 si todos los bits previos de menor orden son iguales a 1 y se habilita la cuenta. La cadena de compuertas AND generan la lógica necesaria para las entradas J y K en cada etapa. El contador puede expandirse a cualquier número de etapas; cada etapa contendrá un flip-flop adicional y una compuerta AND que da una salida de 1 si todas las salidas de los flip-flops previos son 1.

Nótese que los flip-flops se disparan con el flanco negativo del pulso. Esto no es esencial aquí como lo fue en el contador de rizo. El contador podría haberse disparado en el flanco positivo del pulso.

Contador binario creciente-decreciente

En un contador binario sincrónico creciente-decreciente el flip-flop en la posición de menor orden se complementa con cada pulso. Un flip-flop en cualquier otra posición se complementa con un pulso siempre y cuando todos los bits de menor orden sean iguales a cero. Por ejemplo, si el estado presente de un contador binario de 4 bits creciente-decreciente es $A_4A_3A_2A_1 = 1100$, la cuenta siguiente será 1011. A_1 siempre se complementa. A_2 se complementa porque el estado presente de $A_1 = 0$. A_3 se complementa porque el estado presente de $A_2A_1 = 00$. Pero A_4 no se complementa porque el estado presente de $A_3A_2A_1 = 100$, el cual no es una condición de solo ceros.

Un contador binario creciente-decreciente puede ser construido como se muestra en la Figura 7-17, excepto que las entradas de las compuertas AND deben venir de las salidas complementadas de Q' y no de las salidas

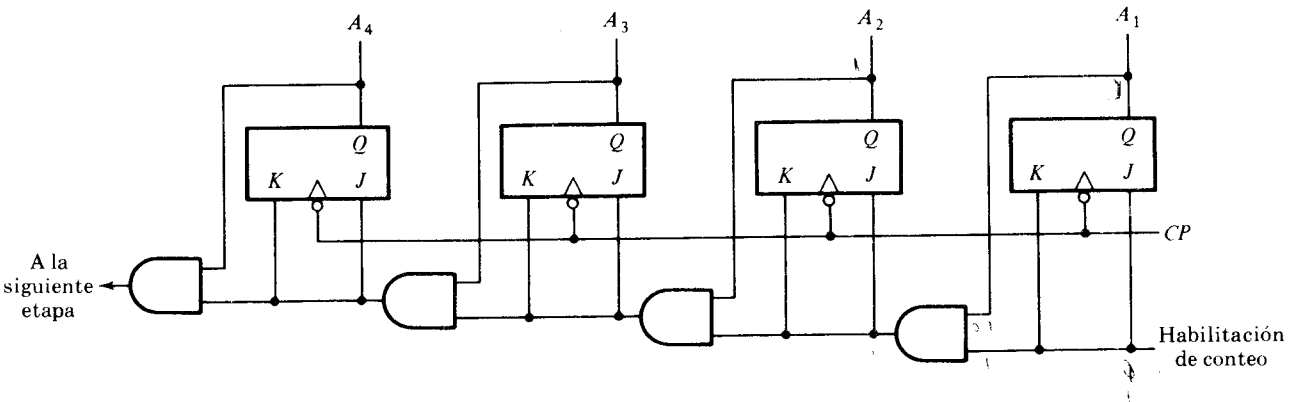


Figura 7-17 Contador binario sincrónico de 4 bits

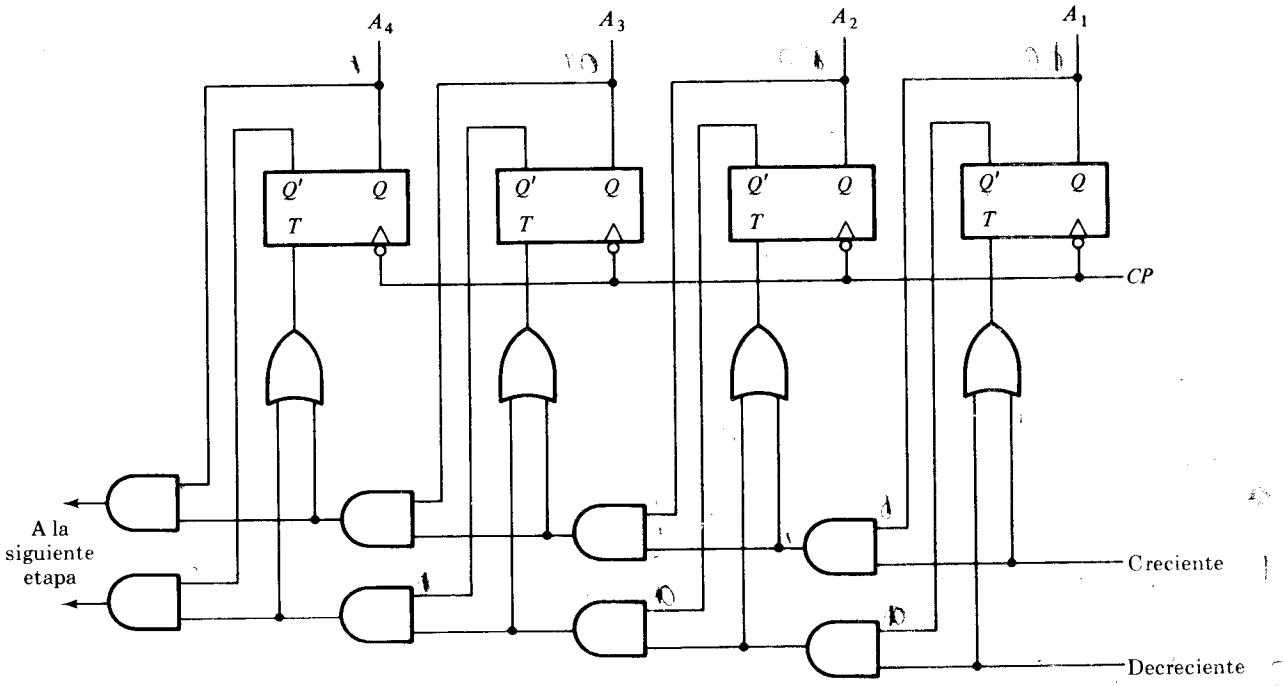


Figura 7-18 Contador binario creciente-decreciente de 4 bits

normales Q de los flip-flops previos. Las dos operaciones se pueden combinar en un circuito. Un contador binario capaz de contar hacia arriba o hacia abajo se muestra en la Figura 7-18. Los flip-flops T empleados en este circuito pueden considerarse como flip-flops JK con los terminales J y K unidos entre sí. Cuando la entrada del control *creciente* es 1, el circuito cuenta hacia arriba, ya que las entradas T se determinan a partir de los valores previos de las salidas normales en Q. Cuando la entrada del control *decreciente* es 1, el circuito contará hacia abajo, ya que las salidas complementadas Q' determinan los estados de las entradas T. Cuando ambas señales *creciente* y *decreciente* son 0, el registro no cambia de estado pero permanece en la misma cuenta.

Contador BDC

Un contador BDC cuenta en binario decimal codificado desde 0000 hasta 1001 y de vuelta a 0000. Debido al regreso a 0 después de la cuenta de 9, un contador BDC no tiene un patrón regular como el contador binario directo. Para diseñar el circuito de un contador sincrónico BDC es necesario pasar por un procedimiento de diseño como el discutido en la Sección 6-8.

La secuencia de cuenta de un contador BDC se da en la Tabla 7-5. La excitación para los flip-flops T se obtienen de la secuencia de cuenta. Una salida y se muestra también en la tabla. Esta salida es igual a 1 cuando el contador de estado presente es 1001. De esta manera, y puede habilitar la cuenta de la siguiente década de mayor orden mientras que el mismo pulso cambia la presente década de 1001 a 0000.

Las funciones de entrada del flip-flop de la tabla de excitación pueden ser simplificadas por medio de los mapas. Los estados sin usar para los términos mínimos 10 a 15 se toman como términos de no importa. Las funciones simplificadas se listan a continuación:

$$TQ_1 = 1$$
$$TQ_2 = Q_8'Q_1$$
$$TQ_4 = Q_2Q_1$$
$$TQ_8 = Q_8Q_1 + Q_4Q_2Q_1$$
$$y = Q_8Q_1$$

El circuito puede dibujarse fácilmente con cuatro flip-flops T, cinco compuertas AND y una compuerta OR.

Los contadores sincrónicos BDC pueden conectarse en cascada para formar un contador para los números decimales de cualquier longitud. La conexión en cascada se hace como en la Figura 7-16 excepto que la salida y debe ser conectada a la entrada de cuenta de la década siguiente de mayor orden.

Tabla 7-5 Tabla de excitación para un contador BDC

Secuencia de cuenta				Entradas del flip-flop				Arrastre de salida
Q_8	Q_4	Q_2	Q_1	TQ_8	TQ_4	TQ_2	TQ_1	y
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	1	0
0	0	1	0	0	0	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	0	0	0	1	0
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1	0
1	0	0	1	1	0	0	1	1

Contador binario con carga en paralelo

Los contadores usados en los sistemas digitales a menudo requieren una condición de carga en paralelo para trasferir un número binario inicial antes de la operación de conteo. La Figura 7-19 muestra el diagrama lógico de un registro que tiene una característica de carga en paralelo y puede operar también como un contador.* La entrada de control de carga, cuando es igual a 1, inhabilita la secuencia de cuenta y causa la trasferencia de datos I_1 hasta I_4 a los flip-flops A_1 hasta A_4 respectivamente. Si la entrada de carga es 0 y la entrada del control de cuenta es 1, el circuito opera como un contador. Los pulsos de reloj causan entonces cambios del estado de los flip-flops de acuerdo a la secuencia de cuenta binaria. Si ambas entradas de control son 0, los pulsos de reloj no cambian el estado del registro.

El terminal de salida del arrastre se convierte en 1 si todos los flip-flops son iguales a 1 mientras se habilita la entrada de cuenta. Esta es una condición para complementar los flip-flops que almacenan el bit siguiente de mayor orden. Esta salida es útil para expandir el contador a más de cuatro bits. La velocidad del contador se aumenta si se genera el arrastre directamente de las entradas de todos los flip-flops en vez de ir a través de una cadena de compuertas AND. De manera similar, cada flip-flop se asocia con una compuerta AND que recibe todas las salidas de los flip-flops anteriores directamente para determinar cuándo el flip-flop debe ser complementado.

La operación del contador se resume en la Tabla 7-6. Las cuatro entradas de control: borrado, CP, carga y cuenta determinan el siguiente estado de salida. La entrada de borrado es asincrónica y cuando ésta es 0, causará que el contador sea puesto a cero, independientemente de la presencia de los pulsos de reloj de otras entradas. Esto se indica en la

*Esto es similar pero no idéntico al CI tipo 74161.

tabla por medio de las entradas X, las cuales simbolizan las condiciones de no importa para las otras entradas, bien sea que su valor sea 0 ó 1. La entrada de borrado debe ir al estado de 1 para las operaciones temporizadas listadas en las siguientes tres entradas en la tabla. Con las entradas de carga y cuenta iguales a 0, las salidas no cambian bien sea que se aplique un pulso en el terminal CP o no. Una entrada de carga de 1 causa una trasferencia de las entradas I_1 a I_4 al registro durante el flanco positivo de un pulso de entrada. La información de entrada se carga a un registro a pesar del valor del terminal de cuenta, porque la entrada de cuenta se inhibe cuando el terminal de carga es 1. Si el terminal de cuenta se mantiene en 0, la entrada de cuenta controla la operación del contador. Las salidas cambian a la siguiente cuenta binaria, en la transición del flanco positivo de cada pulso de reloj, pero no ocurre ningún cambio de estado si la entrada de cuenta es 0.

El contador de 4 bits mostrado en la Figura 7-19 puede encapsularse en un CI. Se necesitan dos CI para la construcción de un contador de 8 bits: cuatro CI para un contador de 16 bits y así sucesivamente. El arrastre de salida de un CI debe ser conectado al terminal de cuenta del CI que almacena los cuatro bits siguientes de mayor orden del contador.

Los contadores con la característica de carga en paralelo que tienen un número específico de bits son muy útiles en el diseño de los sistemas digitales. Más tarde se tratarán como registros con carga y características de incremento. La función de *incremento* es una operación que agrega 1 al contenido presente del registro. Al habilitar el control de cuenta durante el período de un pulso de reloj, el contenido del registro se puede incrementar en 1.

Un contador con carga en paralelo puede ser usado para generar cualquier número deseable de secuencias de cuenta. Un contador de N módulos (abreviado en inglés mod N) es un contador que pasa por una secuencia repetida de N cuentas. Por ejemplo, un contador binario de 4 bits es un contador de 16 módulos (mod-16 counter). Un contador BDC es un contador de 10 módulos (mod-10 counter). En algunas aplicaciones, se puede no estar interesado con los N estados particulares que usa el contador de N módulos. Si este es el caso, entonces el contador con carga en paralelo puede usarse para construir cualquier contador de N módulos, siendo N cualquier valor escogido. Esto se explica en el siguiente ejemplo.

EJEMPLO 7-4: Construir un contador de 6 módulos usando el circuito MSI especificado en la Figura 7-19.

La Figura 7-20 muestra cuatro maneras en las cuales un contador con carga en paralelo puede usarse, para generar una secuencia de seis cuentas. En cada caso el control de cuenta se lleva a 1 para habilitar la cuenta por medio de los pulsos en la entrada CP. Se usa también el hecho de que el control de carga inhibe la cuenta y que la operación de borrado es independiente de otras entradas de control.

La compuerta AND en la Figura 7-20(a) detecta la ocurrencia del estado 0101 en la salida. Cuando el contador está en este estado, la entrada de carga es habilitada y todos los ceros de entrada

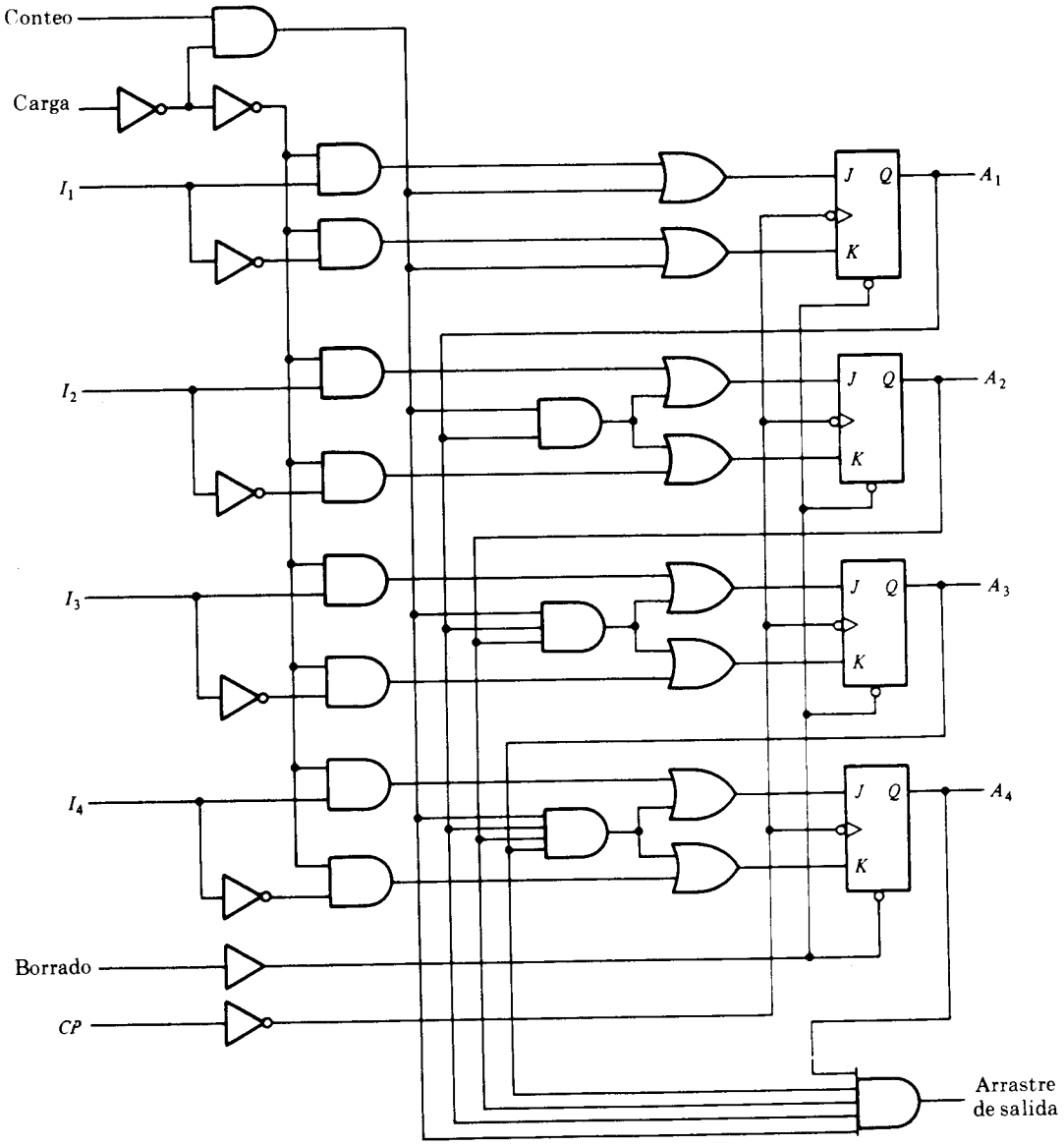


Figura 7-19 Contador binario de 4 bits con carga en paralelo.

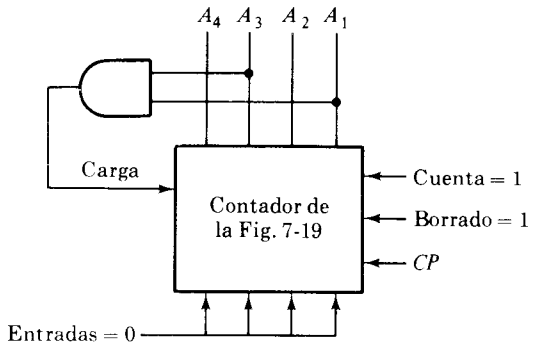
Tabla 7-6 Tabla de función para el contador de la Figura 7-9

Borrado	*CP*	Carga	Conteo	Función
0	*X*	*X*	*X*	Borrar a 0
1	*X*	0	0	No cambiar
1	↑	1	*X*	Cargar entradas
1	↑	0	1	Contar siguiente estado binario

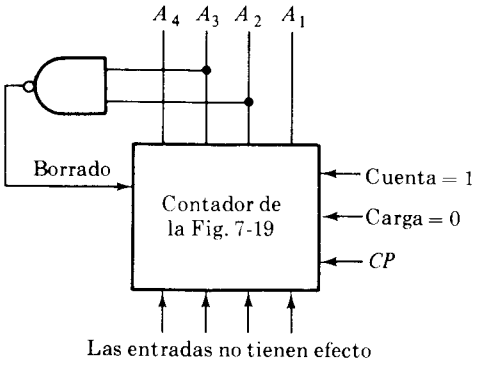
se cargan al registro. Así, el contador pasa por los estados binarios 0, 1, 2, 3, 4 y 5 para regresar luego a cero. Esto produce una secuencia de seis cuentas.

La entrada de borrado del registro es asincrónica es decir, que no depende del reloj. En la Figura 7-20(b), la compuerta NAND detecta la cuenta de 0110, pero tan pronto ocurra esta cuenta, el registro se borra. La cuenta 0110 tiene oportunidad de permanecer por algún tiempo porque el registro va inmediatamente a cero. Un pico momentáneo ocurre en la salida A_2 cuando la cuenta va de 0101 a 0110 e inmediatamente a 0000. Este pico momentáneo puede ser indeseable y por ello no se recomienda esta configuración. Si el contador tiene una entrada de borrado sincrónica, es posible borrar el contador con el reloj después de ocurrir la cuenta 0101.

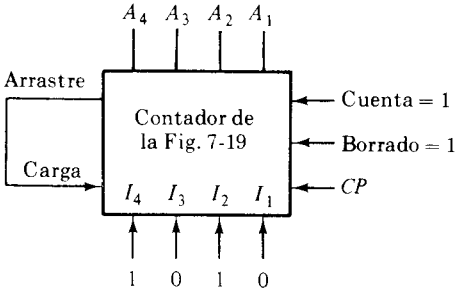
En vez de usar las primeras seis cuentas, se puede desear escoger las últimas seis cuentas desde 10 hasta 15. En este caso es posible tomar ventaja del arrastre de salida para cargar un



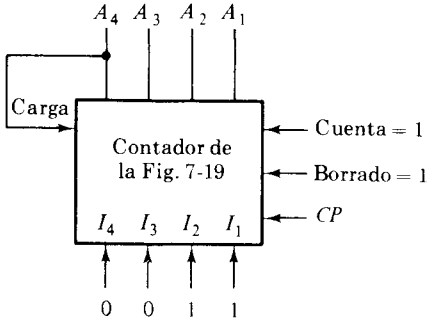
(a) Estados binarios 0, 1, 2, 3, 4, 5.



(b) Estados binarios 0, 1, 2, 3, 4, 5.



(c) Estados binarios 10, 11, 12, 13, 14, 15.



(d) Estados binarios 3, 4, 5, 6, 7, 8.

Figura 7-20 Cuatro maneras de configurar un contador de 6 módulos usando un contador con carga en paralelo

número en el registro. En la Figura 7-20(c), el contador comienza con la cuenta 1010 y continúa hasta 1111. El arrastre de salida generado durante el último estado estable habilita el control de carga, el cual carga entonces la entrada que se establece a 1010.

Es posible también escoger cualquier contador intermedio de seis estados. El contador de 6 módulos de la Figura 7-20(d) pasa por la secuencia de cuenta 3, 4, 5, 6, 7 y 8. Cuando se logra la última cuenta 1000, la salida A_4 va a 1 y se habilita el control de carga. Esto carga al registro el valor 0011 y la cuenta binaria continúa a partir de este estado.

7-6 SECUENCIAS DE TIEMPO

La secuencia de las operaciones en un sistema digital se produce en la unidad de control. La unidad de control que superviza las operaciones en un sistema digital consistiría normalmente en señales de tiempo que determinan la secuencia de tiempo en la cual se ejecutan las operaciones. Las secuencias de tiempo en la unidad de control pueden generarse fácilmente por medio de contadores o registros de desplazamiento. Esta sección demuestra el uso de estas funciones MSI en la generación de señales de tiempo para la unidad de control.

Generación de un tiempo de palabra

Primero, se muestra un circuito que genera la señal de tiempo requerida para un modo de operación en serie. La trasferencia en serie de la información fue discutida en la Sección 7-3, con un ejemplo ilustrado en la Figura 7-8. La unidad de control en un computador en serie debe generar una *señal de tiempo* de palabra que permanezca por un número de pulsos iguales al número de bits en los registros de desplazamiento. La señal de tiempo de palabra puede ser generada por medio de un contador que cuenta el número requerido de pulsos.

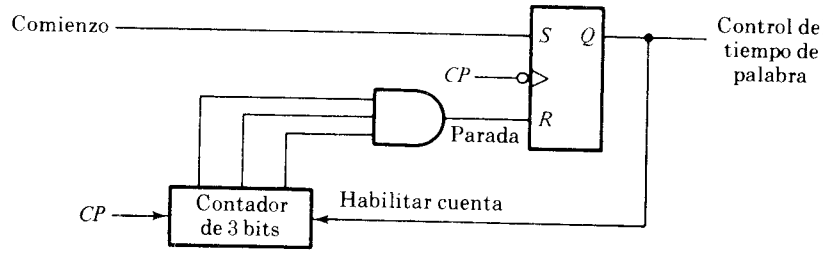
Asúmase que una señal de tiempo de palabra que va a ser generada debe permanecer por un período de ocho pulsos. La Figura 7-2(a) muestra un circuito contador que realiza esta tarea. Inicialmente un contador de 3 bits se borra a 0. Una señal de comienzo pondrá a cero el flip-flop Q. La salida de este flip-flop suministra el control de tiempo de palabra y también habilita el contador. Después de una cuenta de ocho pulsos, el flip-flop se pone a cero y Q va a 0. El diagrama de tiempo de la Figura 7-21(b) demuestra la operación del circuito. La señal de comienzo se sincroniza con el reloj y permanece por un período de un pulso de reloj. Después de que Q se ponga a 1, el contador comienza a contar los pulsos de reloj. Cuando el contador alcanza la cuenta de 7 (binario 111), enviará una señal de parada a la entrada de puesta a cero del flip-flop. La señal de parada se convierte en 1 después de la transición por flanco negativo del pulso 7. El siguiente pulso de reloj cambia el contador al estado 000 y también borra a Q. Ahora el contador se habilita y el tiempo de palabra permanece en 0. Nótese que el control de tiempo de palabra permanece por un período de ocho pulsos.

Nótese también que la señal de parada en este circuito puede usarse para comenzar otro control de cuenta de palabra en otro circuito justamente cuando se usa la señal de comienzo en este circuito.

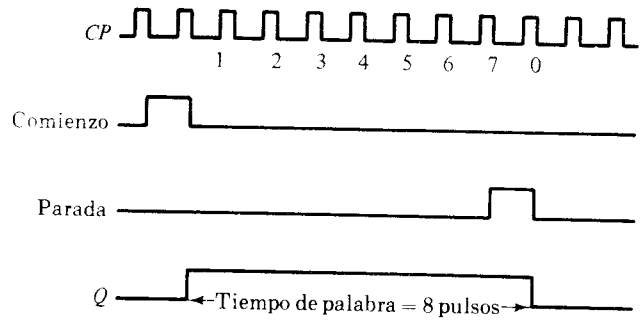
Señales de tiempo

En un modo paralelo de operación, un solo pulso de reloj puede especificar el tiempo durante el cual puede ejecutar la operación. La unidad de control en un sistema digital que opera en el modo en paralelo debe generar señales de tiempo que permanecen por un solo período de pulso, pero estas señales de tiempo deben distinguirse entre sí.

Las señales de tiempo que controlan la secuencia de operaciones en un sistema digital pueden ser generadas con un registro de desplazamiento o un contador con un decodificador. Un *contador de anillo* es un registro de desplazamiento circular con sólo un flip-flop que se pone a uno en un tiempo particular y todos los demás se ponen a cero. El solo bit se desplaza de un flip-flop a otro para producir la secuencia de señales de tiempo. La Figura 7-22(a) muestra un registro de desplazamiento de 4 bits conectados a un contador de anillo. El valor inicial del registro es 1000, lo cual produce la variable T_0. El solo bit se desplaza a la derecha con cada pulso de reloj y circula de nuevo de T_3 a T_0. Cada flip-flop está en el estado de 1, una vez cada cuatro pulsos de reloj y produce una de las cuatro señales de tiem-



(a) Diagrama del circuito



(b) Diagrama de tiempo

Figura 7-21 Generación de un control de tiempo de palabra para operaciones en serie

po mostradas en la Figura 7-22(c). Cada salida se convierte en 1, después de la transición por flanco negativo de un pulso de reloj y permanece en 1 durante el siguiente pulso de reloj.

Las señales de tiempo pueden ser generadas también por habilitación continua de un contador de 2 bits que pasa por cuatro estados diferentes. El decodificador mostrado en la Figura 7-22(b) decodifica los cuatro estados del contador y genera la secuencia requerida de las señales de tiempo.

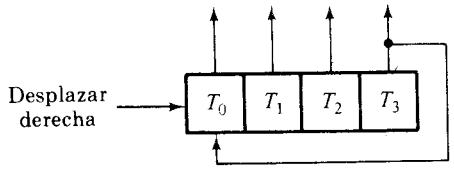
Las señales de tiempo, una vez que se habiliten por el pulso de reloj, suministrarán pulsos de reloj de múltiple fase. Por ejemplo, si T_0 se aplica con CP a una compuerta AND, la salida de la compuerta genera los pulsos de reloj de un cuarto de frecuencia de los pulsos de reloj maestros. Los pulsos de reloj de múltiple fase pueden ser usados para controlar diferentes registros con diferentes estados de tiempo.

Para generar 2^n señales de tiempo, se necesita o un registro de desplazamiento con 2^n flip-flops o un contador de n bits con un codificador de n a 2^n líneas. Por ejemplo, 16 señales de tiempo pueden ser generadas con un registro de desplazamiento de 16 bits conectados a un contador de anillo o con un contador de 4 bits y un decodificador de 4 a 16 líneas. En el primer caso, se necesitan 16 flip-flops. En el segundo caso, se necesitan cuatro flip-flops y 16 compuertas AND de 4 entradas para el decodificador. Es posible generar las señales de tiempo con una combinación de registro de desplazamiento y un decodificador. De esta manera, el número de flip-flops es menor que en un contador de anillo y el decodificador requiere solamente compuertas de 2 entradas. Esta combinación se llama algunas veces un *contador Johnson*.

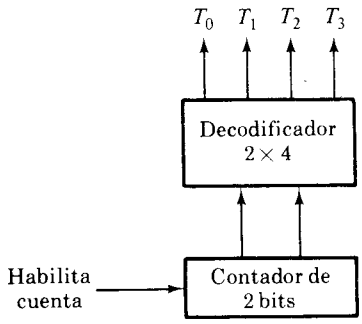
Contador Johnson

Un contador de anillo de k-bits circula un solo bit por los flip-flops para suministrar k estados distinguibles. El número de estados pueden doblarse si el registro de desplazamiento se conecta como un contador de anillo de *final conmutado* (switch-tail ring counter). Un contador de anillo de de final conmutado es un registro de desplazamiento circular con la salida complementada del último flip-flop conectado a la entrada del primer flip-flop. La Figura 7-23(a) muestra tal registro de desplazamiento. La conexión circular se hace de la salida complementada del flip-flop del extremo derecho a la entrada del flip-flop del extremo izquierdo. El registro desplaza su contenido una vez a la derecha con cada pulso de reloj y al mismo tiempo, el valor complementado del flip-flop E se trasfiere al flip-flop A. Comenzando de un estado de borrado, el contador de anillo de final conmutado pasa por una secuencia de ocho estados de la manera listada en la Figura 7-23(b). En general un contador de anillo de final conmutado de k-bits pasará a través de una secuencia de $2k$ estados. Comenzando en 0, cada operación de desplazamiento inyecta unos por la izquierda hasta que el registro se llene de sólo unos. En las secuencias siguientes, se inyectan ceros por la izquierda hasta que el registro se llene con 0.

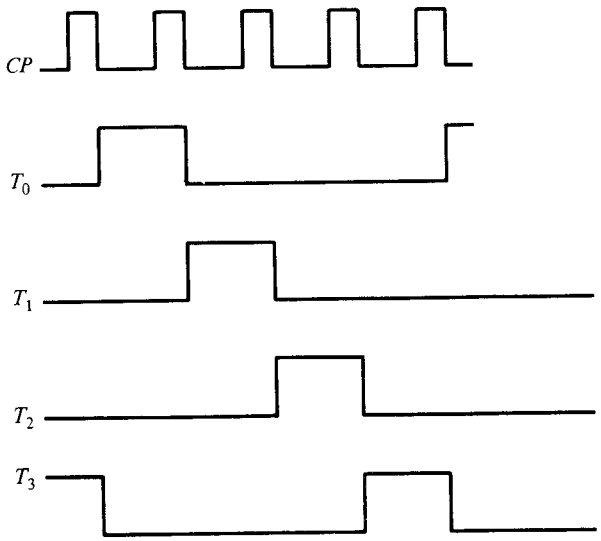
Un contador Johnson es un contador de anillo de final conmutado de k-bits con $2k$ compuertas decodificadoras para suministrar salidas para $2k$ señales de tiempo. Las compuertas decodificadoras no se muestran en



(a) Contador de anillo (valór inicial = 1000)

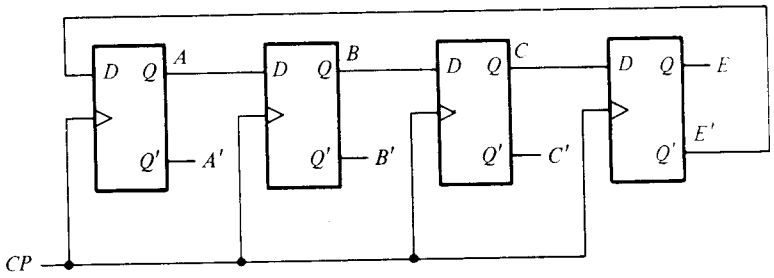


(b) Contador y decodificador



(c) Secuencia de cuatro señales de tiempo

Figura 7-22 Generación de señales de tiempo



(a) Contador de anillo de final conmutado de 4 estados

Número de la secuencia	Salidas de flip-flops A	B	C	E	Compuerta AND requerida para la salida
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Secuencia de conteo y decodificador requerida

Figura 7-23 Construcción de un contador Johnson

la Figura 7-23 pero se especifican en la última columna de la tabla. Las ocho compuertas AND listadas en la tabla, una vez conectadas al circuito, completarán la construcción del contador Johnson. Como cada compuerta se habilita durante una secuencia de estado particular, las salidas de las compuertas generarán ocho secuencias de tiempo en cadencia.

La decodificación de un contador de anillo de final conmutado de k-bits para obtener $2k$ secuencias de tiempo sigue un patrón regular. El estado de sólo 0 se decodifica tomando las salidas normales de los flip-flops de los dos extremos. Todos los otros estados se decodifican de un patrón adyacente de 1, 0 ó 0, 1 en la secuencia. Por ejemplo la secuencia 7 tiene un patrón adyacente 0, 1 en los flip-flops B y C. La salida decodificada se obtiene entonces tomando el complemento de B y la salida normal de C, ó $B'C$.

Una desventaja del circuito en la Figura 7-23(a) es que, si se encuentra en un estado desconocido, persistirá en pasar de un estado no válido a otro y nunca encontrará un camino a un estado válido. Esta dificultad puede ser corregida modificando el circuito para evitar esta condición no deseable. Un procedimiento de corrección es desconectar la salida del flip-flop B que va a la entrada D del flip-flop C, y a cambio habilitar la entrada del flip-flop C por medio de la función:*

*Esta es la manera que se hace en el CI tipo 4022.

$$DC = (A + C)B$$

donde DC es la función del flip-flop para la entrada D del flip-flop C.

Los contadores Johnson pueden ser construidos con cualquier número de secuencias de tiempo. El número de flip-flops necesarios es la mitad del número de señales de tiempo. El número de compuertas decodificadoras es igual al número de señales de tiempo y solamente se emplean compuertas de 2 entradas.

7-7 LA UNIDAD DE MEMORIA

Los registros de un computador digital pueden ser clasificados del tipo operacional o de almacenamiento. Un circuito *operacional* es capaz de acumular información binaria en sus flip-flops y además tiene compuertas combinacionales capaces de realizar tareas de procesamiento de datos. Un registro de *almacenamiento* se usa solamente para el almacenamiento temporal de la información binaria. Esta información no puede ser alterada cuando se trasfiere hacia adentro y afuera del registro. Una *unidad de memoria* es una colección de registros de almacenamiento conjuntamente con los circuitos asociados necesarios para trasferir información hacia adentro y afuera de los registros. Los registros de almacenamiento en una unidad de memoria se llaman *registros de memoria*.

La mayoría de los registros en un computador digital son registros de memoria, a los cuales se trasfiere la información para almacenamiento y de los cuales se obtiene la información necesaria para el procesamiento. Comparativamente se encuentran pocos registros operacionales en la unidad procesadora. Cuando se lleva a cabo el procesamiento de datos, la información de los registros seleccionados en la unidad de memoria se trasfiere primero a los registros operacionales en la unidad procesadora. Los resultados intermedios y finales que se obtienen en los registros operacionales se trasfieren de nuevo a los registros de memoria seleccionados. De manera similar, la información binaria recibida de los elementos de entrada se almacena primero en los registros de memoria. La información trasferida a los elementos de salida se toma de los registros en la unidad de memoria.

El componente que forma las celdas binarias de los registros en una unidad de memoria debe tener ciertas propiedades básicas, de las cuales las más importantes son: (1) debe tener una propiedad dependiente de dos estados para la representación binaria. (2) debe ser pequeño en tamaño. (3) el costo por bit de almacenamiento debe ser lo más bajo posible. (4) el tiempo de acceso al registro de memoria debe ser razonablemente rápido. Ejemplos de componentes de unidad de memoria son los núcleos magnéticos, los CI semiconductores y las superficies magnéticas de las cintas, tambores y discos.

Una unidad de memoria almacena información binaria en grupos llamados *palabras*, cada palabra se almacena en un registro de memoria. Una palabra en la memoria es una entidad de n bits que se mueven hacia adentro y afuera del almacenamiento como una unidad. Una palabra de memoria puede representar un operando, una instrucción, o un grupo de caracteres

alfanuméricos o cualquier información codificada binariamente. La comunicación entre una unidad de memoria y lo que la rodea se logra por medio de dos señales de control y dos registros externos. Las señales de control especifican la dirección de la trasferencia requerida, esto es, cuando una palabra debe ser acumulada en un registro de memoria o cuando una palabra almacenada previamente debe ser trasferida hacia afuera del registro de memoria. Un registro externo especifica el registro de memoria particular escogido entre los miles disponibles; el otro especifica la configuración de bits particular de la palabra en cuestión. Las señales de control y los registros se muestran en el diagrama de bloque de la Figura 7-24.

El *registro de direcciones* de memoria especifica la palabra de memoria seleccionada. A cada palabra en la memoria se le asigna un número de identificación comenzando desde 0 hasta el número máximo de palabras disponible. Para comunicarse con una palabra de memoria específica, su número de localización o *dirección* se trasfiere al registro de direcciones. Los circuitos internos de la unidad de memoria aceptan esta dirección del registro y abren los caminos necesarios para seleccionar la palabra buscada. Un registro de dirección con n bits puede especificar hasta 2^n palabras de memoria. Las unidades de memoria del computador pueden tener un rango entre 1.024 palabras que necesitan un registro de direcciones de 10 bits, hasta $1.048.576 = 2^{20}$ palabras que necesitan un registro de direcciones de 20 bits.

Las dos señales de control aplicadas a la unidad de memoria se llaman de *lectura* y *escritura*. Una señal de escritura especifica una función de trasferencia entrante; una señal de lectura especifica una función de

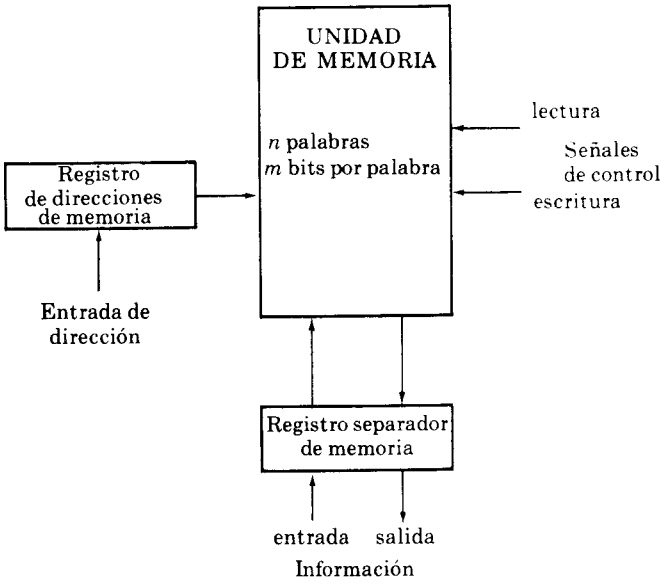


Figura 7-24 Diagrama de bloque de una unidad de memoria mostrando su comunicación con lo que lo rodea

trasferencia saliente. Cada una es referenciada por la unidad de memoria. Después de aceptar una de las señales, los circuitos de control interno dentro de la unidad de memoria suministran la función deseada. Cierto tipo de unidades de almacenamiento, debido a las características de sus componentes, destruyen la información almacenada en una celda cuando se lea el bit de ella. Este tipo de unidad se dice que es una memoria de lectura destructible en oposición a una memoria no destructible donde la información permanece en la celda después de haberse leído. En cada caso, la información primaria se destruye cuando se escribe la nueva información. La secuencia del control interno en una memoria de lectura destructible debe proveer señales de control que puedan causar que la palabra sea restaurada en sus celdas binarias si la aplicación requiere de una función no destructiva.

La información trasferida hacia adentro y afuera de los registros en la memoria y al ambiente externo, se comunica a través de un registro común llamado (buffer register) *registro separador* de memoria (otros nombres son *registro de información* y *registro de almacenamiento*). Cuando la unidad de memoria recibe una señal de control de *escritura*, el control interno interpreta el contenido del registro separador como la configuración de bits de la palabra que se va a almacenar en un registro de memoria. Con una señal de control de *lectura*, el control interno envía la palabra del registro de memoria al registro separador. En cada caso el contenido del registro de direcciones especifica el registro de memoria particular referenciado para escritura o lectura. Por medio de un ejemplo se puede resumir las características de trasferencia de información de una unidad de memoria. Considérese una unidad de memoria de 1.024 palabras con ocho bits por palabra. Para especificar 1.024 palabras, se necesita una dirección de 10 bits, ya que $2^{10} = 1.024$. Por tanto, el registro de direcciones debe contener diez flip-flops. El registro separador debe tener ocho flip-flops para almacenar los contenidos de las palabras trasferidas hacia dentro y afuera de la memoria. La unidad de memoria tiene 1.024 registros con números asignados desde 0 hasta 1.023.

La Figura 7-25 muestra el contenido inicial de tres registros: el registro de direcciones de memoria, (MAR = memory address register) el registro separador de memoria (MBR = memory buffer register) y el registro de memoria direccionado por MAR. Como el número binario equivalente en MAR es el decimal 42, el registro de memoria direccionado por el MAR es uno con un número de dirección 42.

La secuencia de operaciones necesarias para comunicarse con la unidad de memoria para propósitos de trasferir una palabra hacia afuera dirigida al MBR es:

1. Trasferir los bits de dirección de la palabra seleccionada al MAR.
2. Activar la entrada de control de *lectura*.

El resultado de la operación de lectura se ilustra en la Figura 7-26(a). La información binaria almacenada hasta el presente en el registro de memoria 42 se trasfiere al MBR.

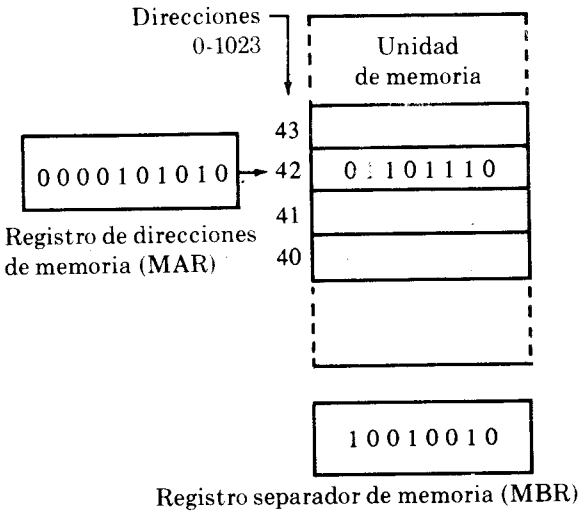


Figura 7-25 Valores iniciales de los registros

La secuencia de operaciones necesarias para almacenar una nueva palabra a la memoria es:

1. Trasferir los bits de dirección de la palabra seleccionada al MAR.
2. Trasferir los bits de datos de la palabra al MBR.
3. Activar la entrada de control de *escritura*.

El resultado de la operación de escritura se ilustra en la Figura 7-26(b). Los bits de datos del MBR se almacenan en el registro de memoria 42.

En el ejemplo anterior, se asume una unidad de memoria con la propiedad de lectura no destructiva. Tales memorias pueden ser construidas con CI semiconductores. Ellas retienen la información en el registro de memoria cuando el registro se catea durante el proceso de lectura de manera que no ocurre pérdida de información. Otro componente usado comúnmente en las unidades de memoria es el núcleo magnético. Un núcleo magnético tiene la característica de tener lecturas destructivas, es decir, pierde la información binaria almacenada durante el proceso de lectura. Ejemplos de memorias de semiconductores y de núcleos magnéticos se presentan en la Sección 7-8.

Debido a la propiedad de lectura destructiva, una memoria de núcleos magnéticos debe tener funciones de control adicionales para reponer la palabra al registro de memoria. Una señal de control de lectura aplicada a una memoria de núcleos magnéticos trasfiere el contenido de la palabra direccionada a un registro externo y al mismo tiempo se borra el registro de memoria. La secuencia de control interno en una memoria de núcleos magnéticos suministra entonces señales apropiadas para causar la recuperación de la palabra en el registro de memoria. La trasferencia de información de una memoria de núcleos magnéticos durante una operación de

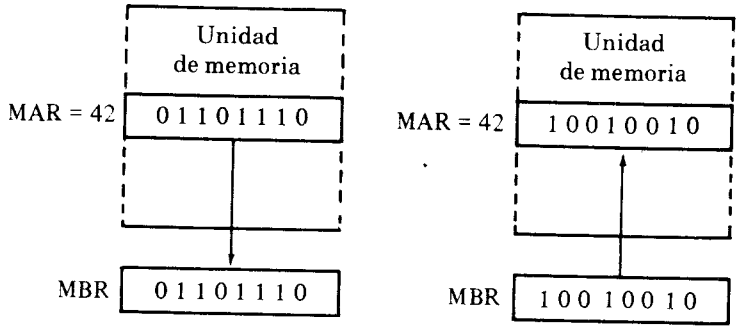


Figura 7-26 Trasferencia de información durante las operaciones de lectura y escritura

lectura se ilustra en la Figura 7-27. Una operación de lectura destructiva trasfiere la palabra seleccionada al MBR pero deja el registro de memoria con puros ceros. La operación de memoria normal requiere que el contenido de la palabra seleccionada permanezca en la memoria después de la operación de lectura. Por tanto, es necesario pasar por una operación de *recuperación* que escribe el valor del MBR en el registro de memoria seleccionada. Durante la operación de recuperación, los contenidos del MAR y el MBR deben permanecer invariables.

Una entrada de control de escritura aplicada a una memoria de núcleos magnéticos causa una trasferencia de información como se muestra en la Figura 7-28. Para trasferir la nueva información a un registro seleccionado, se debe primero borrar la información anterior borrando todos los bits de la palabra a 0. Después de hacer lo anterior, el contenido del MBR se puede trasferir a la palabra seleccionada. El MAR no debe cambiar durante la operación para asegurar que la misma palabra seleccionada que se ha borrado es aquella que recibe la nueva información.

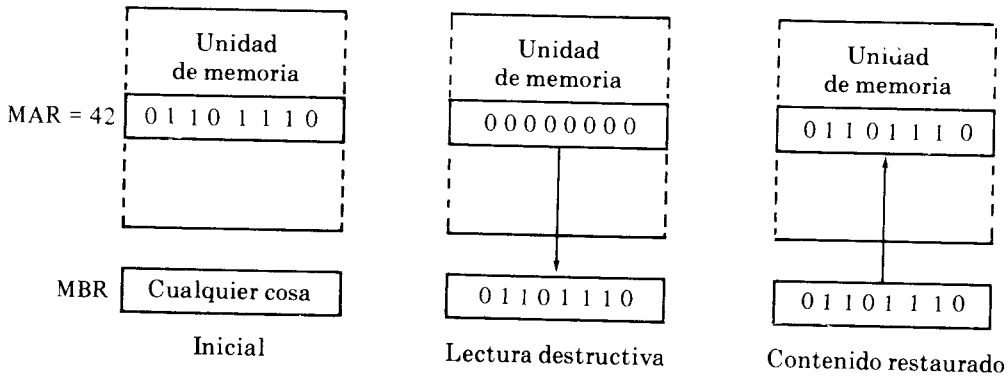


Figura 7-27 Trasferencia de información en una memoria de núcleos magnéticos durante una operación de lectura

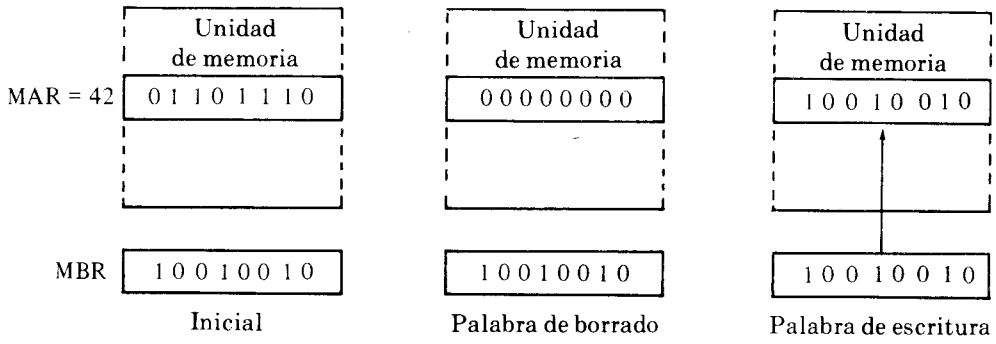


Figura 7-28 Trasferencia de información en una memoria de núcleos magnéticos durante una operación de escritura

Una memoria de núcleo magnético requiere dos medios ciclos para leer o escribir. El tiempo que se toma la memoria para cubrir los dos medios ciclos se llama tiempo de *un ciclo de memoria*.

El modo de acceso de un sistema de memoria se determina por el tipo de componentes usados. En una memoria *de acceso aleatorio*, se debe pensar que los registros están separados en el espacio, con cada registro ocupando un lugar espacial particular en una memoria de núcleos magnéticos. En una memoria de *acceso secuencial*, la información almacenada en algún medio no es accesible inmediatamente pero se obtiene solamente en ciertos intervalos de tiempo. Una unidad de cinta magnética es de este tipo. Cada lugar de la memoria pasa por las cabezas de lectura y escritura a la vez, pero la información se lee solamente cuando se ha logrado la palabra solicitada. El *tiempo de acceso* de una memoria es el tiempo requerido para seleccionar una palabra o en la lectura o en la escritura. En una memoria de acceso aleatorio, el tiempo de acceso es siempre el mismo a pesar del lugar en el espacio particular de la palabra. En una memoria secuencial, el tiempo de acceso depende de la posición de la palabra en el tiempo que se solicita. Si la palabra está justamente emergiendo del almacenamiento en el tiempo que se solicita, el tiempo de acceso es justamente el tiempo necesario para leerla o escribirla. Pero, si la palabra por alguna razón está en la última posición, el tiempo de acceso incluye también el tiempo requerido para que todas las otras palabras se muevan pasando por los terminales. Así, el tiempo de acceso a una memoria secuencial es variable.

Las unidades de memoria cuyos componentes pierden información almacenada con el tiempo o cuando se corta el suministro de energía, se dice que son *volátiles*. Una unidad de memoria de semiconductores es de esta categoría ya que sus celdas binarias necesitan potencia externa para mantener las señales necesarias. En contraste, una unidad de memoria no volátil, tal como un núcleo magnético o un disco magnético, retiene la información almacenada una vez que se haya cortado el suministro de energía. Esto es debido a que la información acumulada en los componentes magnéticos se manifiestan por la dirección de magnetización, la cual se retiene cuando se corta la energía. Una propiedad no volátil es deseable en los computadores digitales porque muchos programas útiles se dejan perma-

nentemente en la unidad de memoria. Cuando se corte el suministro de energía y luego se suministre, los programas almacenados previamente y otra información no se pierden pero continúan acumulados en la memoria.

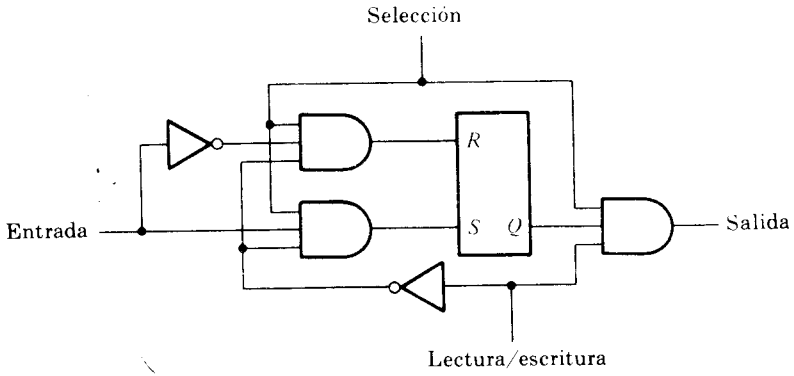
7-8 EJEMPLOS DE MEMORIA DE ACCESO ALEATORIO

La construcción interna de dos tipos diferentes de memorias de acceso aleatorio se presentan en forma de diagramas en esta sección. La primera se construye con flip-flops y compuertas y la segunda con núcleos magnéticos. Para poder incluir toda la unidad de memoria en un diagrama, se debe usar una capacidad de almacenamiento limitado. Por esta razón, las unidades de memoria presentadas aquí tienen una pequeña capacidad de 12 bits arreglados en cuatro palabras de tres bits cada una. Las memorias de acceso aleatorio comerciales pueden tener una capacidad de miles de palabras y cada palabra puede estar en un rango de 8 a 64 bits. La construcción lógica de las unidades de memoria de gran capacidad serían una extensión directa de la configuración mostrada aquí.

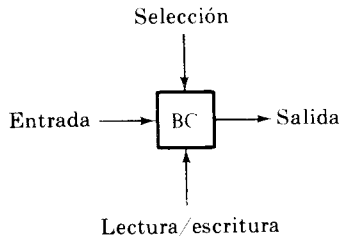
Memoria de circuito integrado

La construcción interna de una memoria de acceso aleatorio de m palabras con n bits por palabra consiste en $m \times n$ celdas de almacenamiento binario y la lógica asociada para seleccionar las palabras individuales. La celda de almacenamiento binario es el bloque básico de construcción de una unidad de memoria. La lógica equivalente de una celda binaria que almacena un bit de información se muestra en la Figura 7-29. Aunque se muestra que la celda incluye compuertas y un flip-flop, internamente se construye con dos transistores que tienen múltiples entradas. Una celda de almacenamiento binario debe ser muy pequeña para poder comprimir tantas celdas como sea posible en la pequeña área disponible en la pastilla de circuito integrado. La celda binaria tiene tres entradas y una salida. La entrada de selección habilita la celda para lectura o escritura. Las entradas de lectura/escritura determinan la operación de la celda cuando esta es seleccionada. Un 1 en la entrada de lectura/escritura forma un camino del flip-flop al terminal de salida. La información en el terminal de entrada se trasfiere al flip-flop cuando el control de lectura/escritura es 0. Nótese que el flip-flop opera sin pulsos de reloj y que su propósito es almacenar la información de bits en la celda binaria.

Las memorias de circuito integrado tienen algunas veces una sola línea para el control de lectura y escritura. Un estado binario en la sola línea especifica una operación de lectura y el otro estado especifica una operación de escritura. Además, se incluyen una o más líneas de habilitación para suministrar medios de seleccionar el CI y para expandir varias pastillas a una unidad de memoria con un gran número de palabras. La construcción lógica de un CI RAM se muestra en la Figura 7-30. Este consiste en 4 palabras de 3 bits cada una para un total de 12 celdas binarias. Los pequeños recuadros marcados BC representan una celda binaria, y las tres entradas y una salida en cada BC son especificadas en el diagrama de la Figura 7-29.



(a) Diagrama lógico



(b) Diagrama de bloque

Figura 7-29 Celda de memoria

Las dos líneas de entrada de direcciones pasan por un decodificador interno de 2 a 4 líneas. El decodificador se habilita con una entrada de habilitación de memoria. Cuando la habilitación de memoria es 0, todas las salidas del decodificador son 0 y ninguna de las palabras en memoria se seleccionan. Con la habilitación de memoria en 1, se selecciona una de las cuatro palabras, dependiendo del valor de las dos líneas de direcciones. Ahora, con el control de lectura/escritura en 1, los bits de la palabra seleccionada pasarán por las 3 compuertas OR hasta los terminales de salida. Las celdas binarias no seleccionadas producen 0 en las entradas de las compuertas OR y no tienen efecto en las salidas. Con el control de lectura/escritura en 0, la información disponible en las líneas de entrada se trasfiere a las celdas binarias de la palabra seleccionada. Las celdas binarias no seleccionadas en las otras palabras son inabilitadas por sus entradas de selección y sus valores previos permanecen sin cambiar. Con el control de habilitación de memoria en 0, el contenido de todas las celdas en la memoria permanece sin cambiar independientemente del valor del control de lectura/escritura.

Un CI RAM se construye internamente con celdas que tienen una característica de OR alambrado. Esto elimina la necesidad de compuertas

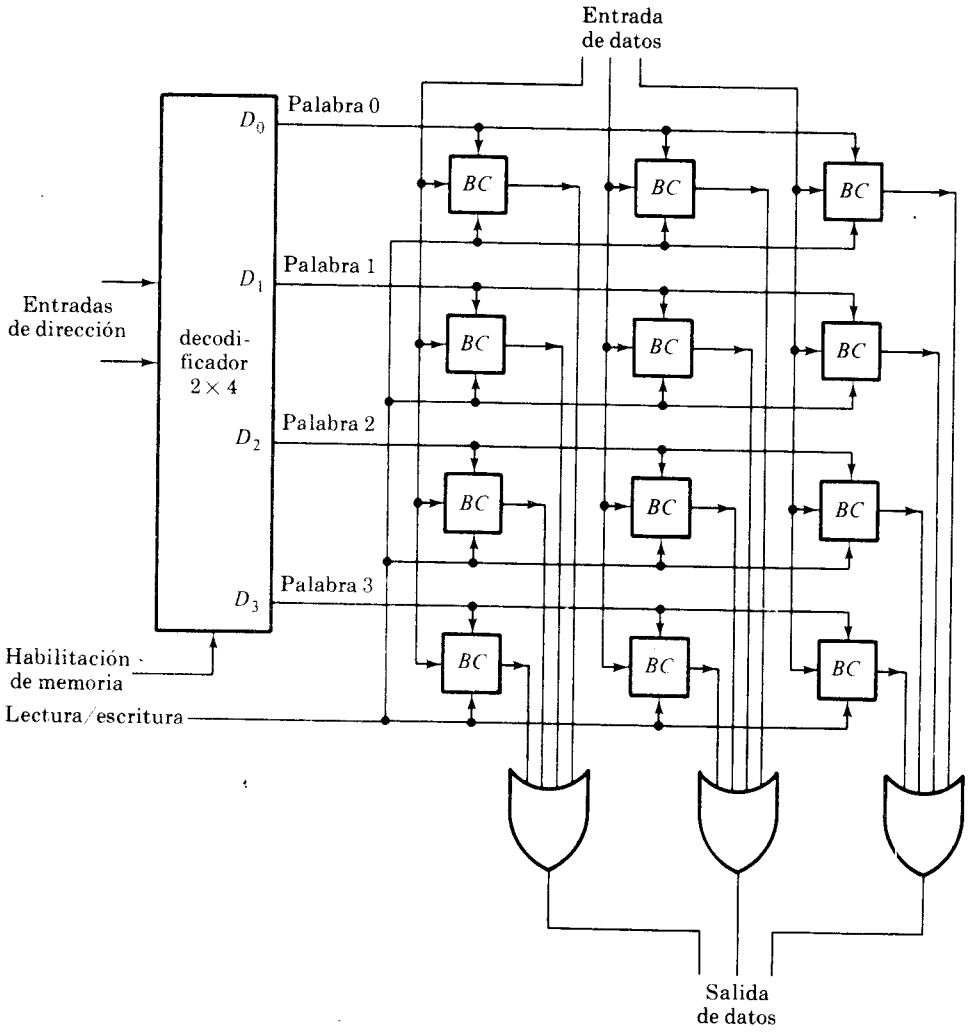


Figura 7-30 Memoria de circuito integrado

OR en el diagrama. Las líneas de salida externas pueden formar también lógica alambrada para facilitar la conexión de dos o más pastillas de CI para formar una unidad de memoria con un gran número de palabras.

Memoria de núcleos magnéticos

Una memoria de núcleos magnéticos usa núcleos magnéticos para almacenar información binaria. Un núcleo magnético es un toroide en forma de rosquilla hecho de material magnético. En contraste con un flip-flop de semiconductores que necesita solamente una cantidad física tal como el voltaje para su operación, un núcleo magnético emplea tres cantidades

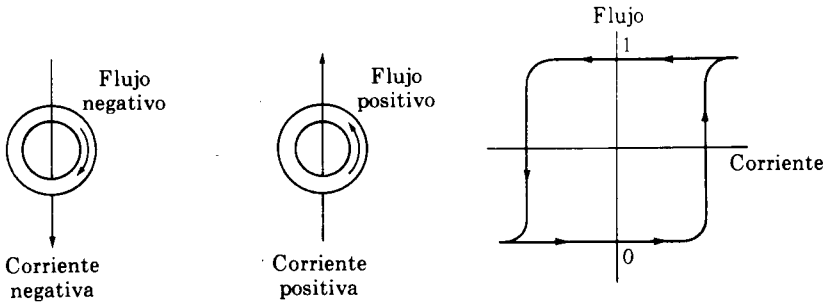
físicas: corriente, flujo magnético y voltaje. La señal que excita el núcleo es un pulso de *corriente* en un alambre que pasa a través del núcleo. La información binaria almacenada se representa por la dirección del *flujo magnético* dentro del núcleo. La información binaria de salida se extrae de un alambre que encadena al núcleo, en la forma de un pulso de *voltaje*.

La propiedad física que hace un núcleo magnético utilizable para almacenamiento binario es su reversión de histérisis, mostrada en la Figura 7-31(c). Esta es un gráfico de la corriente versus el flujo magnético y tiene la forma de una figura cuadrada. Con cero corriente, un flujo que puede ser positivo en dirección (hacia la izquierda) o negativo (hacia la derecha) permanece en el núcleo magnetizado. Se usa una dirección, por ejemplo la magnetización a la izquierda, para representar un 1 y la contraria para representar un 0.

Un pulso de corriente aplicado al alambre que pasa por el núcleo puede cambiar la dirección de magnetización. Como se ve en la Figura 7-31(a), la corriente en dirección hacia abajo produce el flujo en dirección hacia la derecha, causando que el núcleo vaya al estado de 0. La Figura 7-31(b) muestra las direcciones de la corriente y el flujo para almacenar un 1. El cambio que toma el flujo cuando se aplica el pulso de corriente se indica por medio de flechas en el circuito de histéresis.

Leer la información binaria almacenada en el núcleo es muy complicado por el hecho de que el flujo no puede ser detectado cuando no está cambiando. Sin embargo si el flujo está cambiando con respecto al tiempo, este induce un voltaje en el alambre que enlaza el núcleo. Así, la lectura puede llevarse a cabo aplicando una corriente en la dirección negativa como se muestra en la Figura 7-32. Si el núcleo está en el estado 1, la corriente invierte la dirección de magnetización y el cambio resultante de flujo produce un pulso de voltaje en el alambre sensor. Si el núcleo aún está en el estado 0, la corriente negativa deja al núcleo magnetizado en la misma dirección, causando una pequeña distorsión del flujo magnético lo cual producirá un voltaje de salida muy pequeño en el alambre sensor. Nótese que esta es una lectura destructiva ya que la corriente de lectura regresa siempre el núcleo al estado de 0. El valor almacenado previamente se pierde.

La Figura 7-33 muestra la organización de una memoria de núcleos magnéticos que contiene cuatro palabras con tres bits cada una. Compa-



(a) Almacenar 0 (b) Almacenar 1 (c) Figura de histéresis

Figura 7-31 Almacenamiento de un bit en un núcleo magnético

rándola con la unidad de memoria de CI de la Figura 7-30, se nota que la celda binaria es ahora el núcleo magnético y los alambres que lo encadenan. La excitación del núcleo se logra por medio de un pulso de corriente generado por un circuito accionador (DR = Driver). La información de salida pasa por un amplificador sensor (SA = Sense Amplifier) cuyas salidas ponen a uno los flip-flops en el registro separador. Cada núcleo está enlazado por tres alambres. El alambre de palabra es excitado por un accionador de palabras y pasa por tres núcleos de una palabra. El alambre de bits es excitado por un accionador de bit y pasa a través de cuatro núcleos en la misma posición de bit. El alambre sensor enlaza los mismos núcleos que el alambre de bits y se aplica a un amplificador sensor que conforma el pulso de voltaje cuando se lee 1 y rechaza la pequeña distorsión cuando se lee 0.

Durante una operación de lectura, un pulso de corriente accionador de palabra se aplica a los núcleos de la palabra seleccionada por el decodificador. La corriente de lectura está en la dirección negativa (Figura 7-32) y causa que todos los núcleos de la palabra seleccionada vayan al estado de 0 independientemente del estado anterior. Los núcleos que contienen un 1 previamente cambian su flujo e inducen un voltaje al alambre sensor. El flujo de los núcleos que contenía un 0 no cambia. El pulso de voltaje en el alambre sensor de los núcleos con un 1 previo se amplifica en el amplificador sensor y pone a uno el flip-flop correspondiente en el registro separador.

Durante la operación de escritura, el registro separador mantiene la información para ser almacenada en la palabra especificada por el registro de direcciones. Se asume que todos los núcleos de la palabra seleccionada están inicialmente borrados, es decir, todos están en el estado de 0 de tal manera que aquellos que necesiten un 1 deben sufrir un cambio de estado. Un pulso de corriente se genera simultáneamente en el accionador de palabra por el decodificador y en el accionador de bits cuyo flip-flop del registro separador correspondiente contiene un 1. Ambas corrientes están en la dirección positiva, pero su magnitud es solamente la mitad de la necesaria para cambiar el flujo al estado 1. Esta corriente media, en sí misma, es muy pequeña para cambiar la dirección de magnetización. Pero la suma de dos medias corrientes es suficiente para cambiar la dirección de magnetización al estado de 1. Un núcleo cambia al estado de 1 solamente si

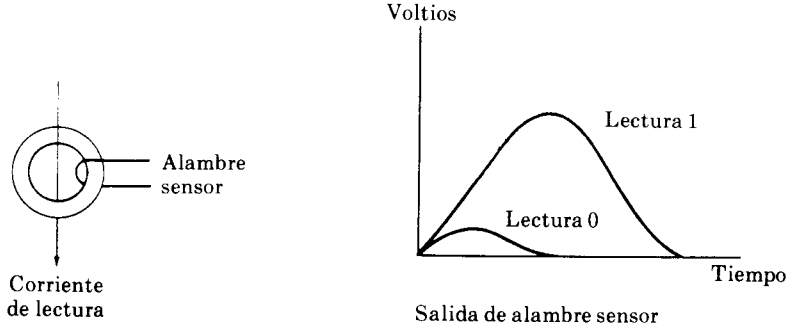


Figura 7-32 Lectura de un bit de un núcleo magnético

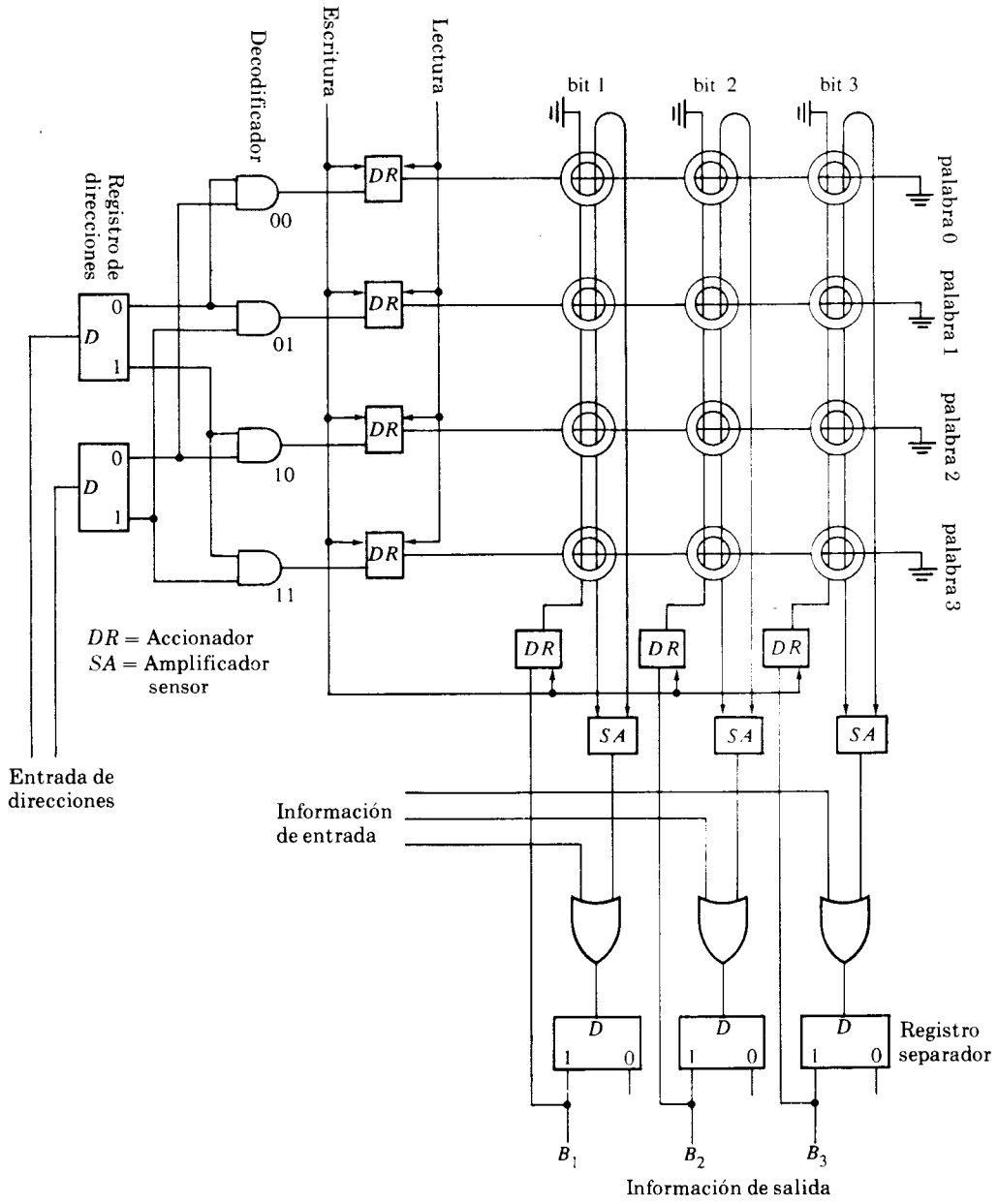


Figura 7-33 Unidad de memoria de núcleos magnéticos

hay una coincidencia de dos medias corrientes de un accionador de palabra y un accionador de bits. La dirección de magnetización de un núcleo no cambia si este recibe solamente media corriente de uno de los accionadores. El resultado es que la magnetización de los núcleos se cambia al estado de 1 solamente si los alambres de palabra y bit se interceptan, esto es, solamente en la palabra seleccionada en la posición de bit en la cual el registro separador es un 1.

Las operaciones de lectura y escritura descritas anteriormente son incompletas, porque la información almacenada en la palabra seleccionada se destruye por el proceso de lectura y la operación de escritura trabaja adecuadamente sólo si los núcleos están borrados inicialmente. Como se menciona en la Sección 7-7 la operación de lectura debe estar seguida por otro ciclo que restaura los valores previamente almacenados en los núcleos. Una operación de escritura está precedida por un ciclo que borra los núcleos de la palabra seleccionada.

La operación de restauración durante el ciclo de lectura es equivalente a la operación de escritura, lo cual, en efecto, escribe la información previamente leída del registro separador de vuelta a la palabra seleccionada. La operación de borrado durante un ciclo de escritura es equivalente a una operación de lectura la cual destruye la información almacenada pero previene la información leída de llegar al registro separador, al inhibir al amplificador sensor. Los ciclos de restauración y borrado se inician normalmente por el control interno de la memoria, de tal manera que la unidad de memoria, parece al mundo exterior, como que tiene una propiedad de lectura no destructiva.

REFERENCIAS

1. *The TTL Data Book for Design Engineers.* Dallas, Texas: Texas Instruments, Inc., 1976.
2. Blakeslee, T. R., *Digital Design with Standard MSI and LSI.* Nueva York: John Wiley & Sons, 1975.
3. Barna A. y D. I. Porat, *Integrated Circuits in Digital Electronics.* Nueva York: John Wiley & Sons, 1973.
4. Taub, H. y D. Schilling, *Digital Integrated Electronics.* Nueva York: McGraw-Hill Book Co., 1977.
5. Grinich, V. H. y H. G. Jackson, *Introduction to Integrated Electronics.* Nueva York: McGraw-Hill Book Co., 1975.
6. Kostopoulos, G. K., *Digital Engineering.* Nueva York: McGraw-Hill Book Co., 1975.
7. Scott, N. R., *Electronic Computer Technology.* Nueva York: McGraw-Hill Book Co., 1970, Capítulo 10.
8. Kline, R. M., *Digital Computer Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977, Capítulo 9.

PROBLEMAS

7-1. El registro de la Figura 7-1 trasfiere la información de entrada a los flip-flops cuando la entrada CP pasa por una transición de flanco negativo. Modifique el circuito de tal manera que la información de entrada se trasfiera al registro cuando un pulso de reloj pasa por una transición de flanco negativo, teniendo en cuenta que la entrada de control de carga es igual al binario 1.

7-2. El registro de la Figura 7-3 carga las entradas durante una transición negativa de un pulso de reloj. ¿Qué cambios internos son necesarios para que las entradas sean cargadas durante el flanco positivo del pulso?

7-3. Verifique el circuito de la Figura 7-5 usando los mapas para simplificar las siguientes ecuaciones de estado.

7-4. Diseñe el circuito secuencial cuya tabla de estado está dada a continuación usando un registro de 2 bits y compuertas combinacionales.

Estado presente		Entrada	Estado siguiente	
A	B	x	A	B
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

7-5. Diseñe un circuito secuencial cuyo diagrama de estado esté dado en la Figura 6-27 usando un registro de 3 bits y una ROM de 16×4.

7-6. El contenido de un registro de desplazamiento de 4 bits es inicialmente 1101. El registro se desplaza seis veces a la derecha, con la entrada en serie siendo 101101. ¿Cuál es el contenido del registro después de cada desplazamiento?

7-7. ¿Cuál es la diferencia entre la trasferencia en serie y en paralelo? ¿Qué tipo de registro se usa en cada caso?

7-8. El registro de desplazamiento bidireccional de 4 bits de la Figura 7-9 se encapsula dentro de una pastilla de CI.

(a) Dibuje un diagrama de bloque de un CI mostrando todas las entradas y salidas.

(b) Dibuje un diagrama de bloque usando tres CI para producir un registro de desplazamiento bidireccional de 12 bits.

7-9. El sumador en serie de la Figura 7-10 usa dos registros de desplazamiento de 4 bits. El registro A retiene el número binario 0101 y el registro B retiene 0111. El flip-flop del arrastre Q se borra inicialmente. Liste los valores binarios en el registro A y el flip-flop Q después de cada desplazamiento.

7-10. ¿Qué cambios son necesarios en el circuito de la Figura 7-11 para convertirlo a un circuito que resta el contenido de B al contenido de A?

7-11. Diseñe un contador en serie; en otras palabras determine el circuito que debe ser incluido externamente con el registro de desplazamiento para poder obtener un contador que opera en serie.

7-12. Dibuje el diagrama de un contador de rizado de 4 bits binario usando flip-flops que se disparan con el flanco positivo.

7-13. Un flip-flop tiene una demora de 20 ns desde el momento en que su entrada *CP* va de 1 a 0 hasta el momento en que se complementa su salida. ¿Cuál es la demora máxima en un contador binario de rizado de 10 bits que usa estos flip-flops? ¿Cuál es la frecuencia máxima con que puede operar el contador confiablemente?

7-14. ¿Cuantos flip-flops deben ser complementados en un contador binario de rizado de 10 bits para alcanzar la siguiente cuenta después de 0111111111?

7-15. Dibuje el diagrama de un contador decreciente binario de rizado de 4 bits usando flip-flops que se disparan en (a) transición de flanco positivo y (b) transición de flanco negativo.

7-16. Dibuje un diagrama de tiempo similar a aquel de la Figura 7-15 para el contador binario de rizado de la Figura 7-12.

7-17. Determine el siguiente estado para cada uno de los seis estados no usados en el contador de rizado BDC de la Figura 7-14. ¿Es el contador autocomenzante?

7-18. El contador de rizado demostrado en la Figura P7-18 usa flip-flops que se disparan en la transición de flanco negativo de la entrada *CP*. Determine la secuencia de cuenta del contador. ¿Es el contador autocomenzante?

7-19. ¿Qué pasa al contador de la Figura 7-18 si ambas entradas creciente y decreciente son iguales a 1 al mismo tiempo? Modifique el circuito de tal manera que cuente hacia arriba si ocurre esta condición.

7-20. Verifique las funciones de entrada del flip-flop del contador BDC sincrónico especificado por la Tabla 7-5. Dibuje el diagrama lógico del contador BDC e incluye una entrada de control de habilitación de cuenta.

7-21. Diseñe un contador BDC sincrónico con flip-flops *JK*.

7-22. Muestre las conexiones externas de cuatro contadores binarios de CI con carga en paralelo (Figura 7-19) para producir un contador binario de 16 bits. Use un diagrama de bloque para cada CI.

7-23. Construya un contador BDC usando un circuito MSI de la Figura 7-19.

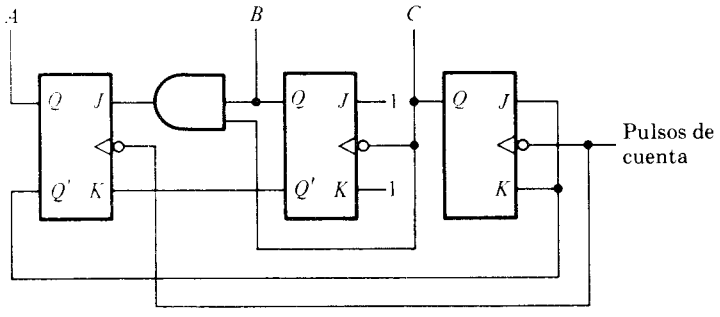


Figura P7-18 Contador de rizado

7-24. Construya un contador de 12 módulos usando el circuito MSI especificado en la Figura 7-19. Dé cuatro alternativas.

7-25. Usando los dos circuitos MSI especificados en la Figura 7-19, construya un contador binario que cuente desde 0 hasta el binario 64.

7-26. Usando la variable de *parada* de la Figura 7-21 como señal de comienzo construya un segundo control de tiempo de palabra que permanezca por un período de 16 pulsos de reloj.

7-27. Demuestre que un contador binario de n bits conectado a un decodificador de n a 2^n líneas es equivalente a un contador de anillo con 2^n flip-flops. Dibuje los diagramas de bloque de ambos circuitos para $n = 3$. ¿Cuántas señales de tiempo se generan?

7-28. Incluya una entrada de habilitación para el decodificador de la Figura 7-22(b) y conéctela a los pulsos de reloj. Dibuje las señales de tiempo que se generen ahora a las salidas del decodificador.

7-29. Complete el diseño del contador Johnson de la Figura 7-23 mostrando las salidas de las ocho señales de tiempo.

7-30. (a) Liste los ocho estados no usados en el contador de anillo de final conmutado de la Figura 7-23. Determine el siguiente estado para cada estado no usado y muestre que, si el circuito se encuentra en un estado inválido, este no regresa a un estado válido. (b) Modifique el circuito como se recomienda en el texto y demuestre que (1) el circuito produce la misma secuencia de estados como la listada en la Figura 7-23(b), y (2) el circuito alcanza un estado válido de cualquiera de los estados no válidos.

7-31. Construya un contador Johnson con diez señales de tiempo.

7-32. (a) La unidad de memoria de la Figura 7-24 tiene una capacidad de 8.192 palabras de 32 bits por palabra. ¿Cuántos flip-flops se necesitan para el registro de dirección de memoria y el registro separador de memoria? (b) ¿Cuántas palabras contendrá la unidad de memoria si el registro de dirección tiene 15 bits?

7-33. Cuando el número de palabras que se van a seleccionar es muy grande, es conveniente usar una celda de almacenamiento binario con dos entradas de selección: una entrada de selección X (horizontal) y una Y (vertical). Ambas X y Y deben ser habilitadas para seleccionar la celda.

(a) Dibuje una celda binaria similar a la de la Figura 7-29 con las entradas de selección X y Y.

(b) Demuestre cómo pueden ser usados dos decodificadores de 4×16 para seleccionar una palabra en una memoria de 256 palabras.

7-34. (a) Dibuje un diagrama de bloque de la memoria de 4×3 de la Figura 7-30, mostrando todas las entradas y salidas. (b) Construya una memoria de 8×3 que usa dos de estas unidades. Use una construcción de diagrama de bloque.

7-35. Se requiere construir una memoria con 256 palabras, 16 bits por palabra organizada como en la Figura 7-33. Los núcleos están disponibles en una matriz de 16 filas y 16 columnas.

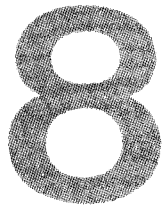
(a) ¿Cuántas matrices se necesitan?

(b) ¿Cuántos flip-flops hay en los registros de dirección y reparación?

(c) ¿Cuántos núcleos reciben corriente durante el ciclo de lectura?

(d) ¿Cuántos núcleos reciben al menos media corriente durante un ciclo de escritura?

Lógica de trasferencia entre registros



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

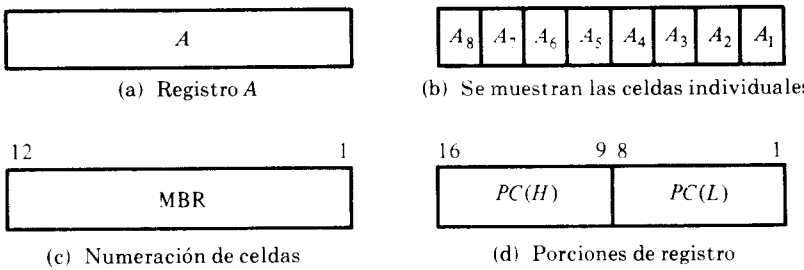


Figura 8-1 Diagrama de bloque de los registros

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.

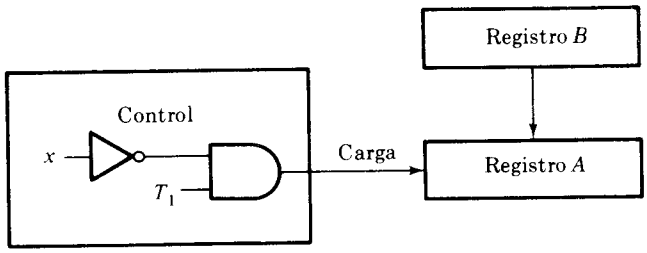


Figura 8-2 Configuración con componentes de la proposición $x'T_1$: $A \leftarrow B$

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:

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

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:

$$T_1: \quad C \leftarrow A$$

$$T_5: \quad C \leftarrow B$$

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

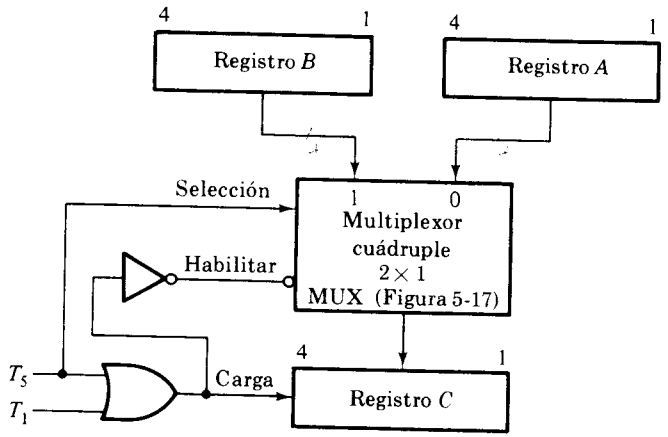


Figura 8-3 Uso de un multiplexor para trasferir información de dos fuentes a un solo destino

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-

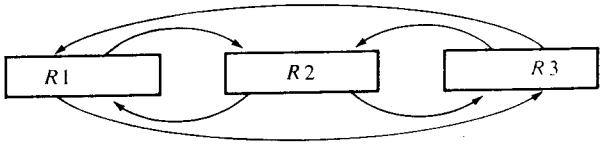


Figura 8-4 Trasferencia entre tres registros

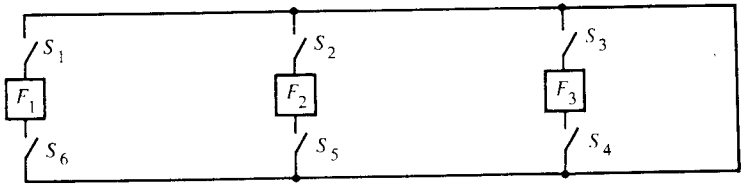


Figura 8-5 Trasferencia a través de una línea común

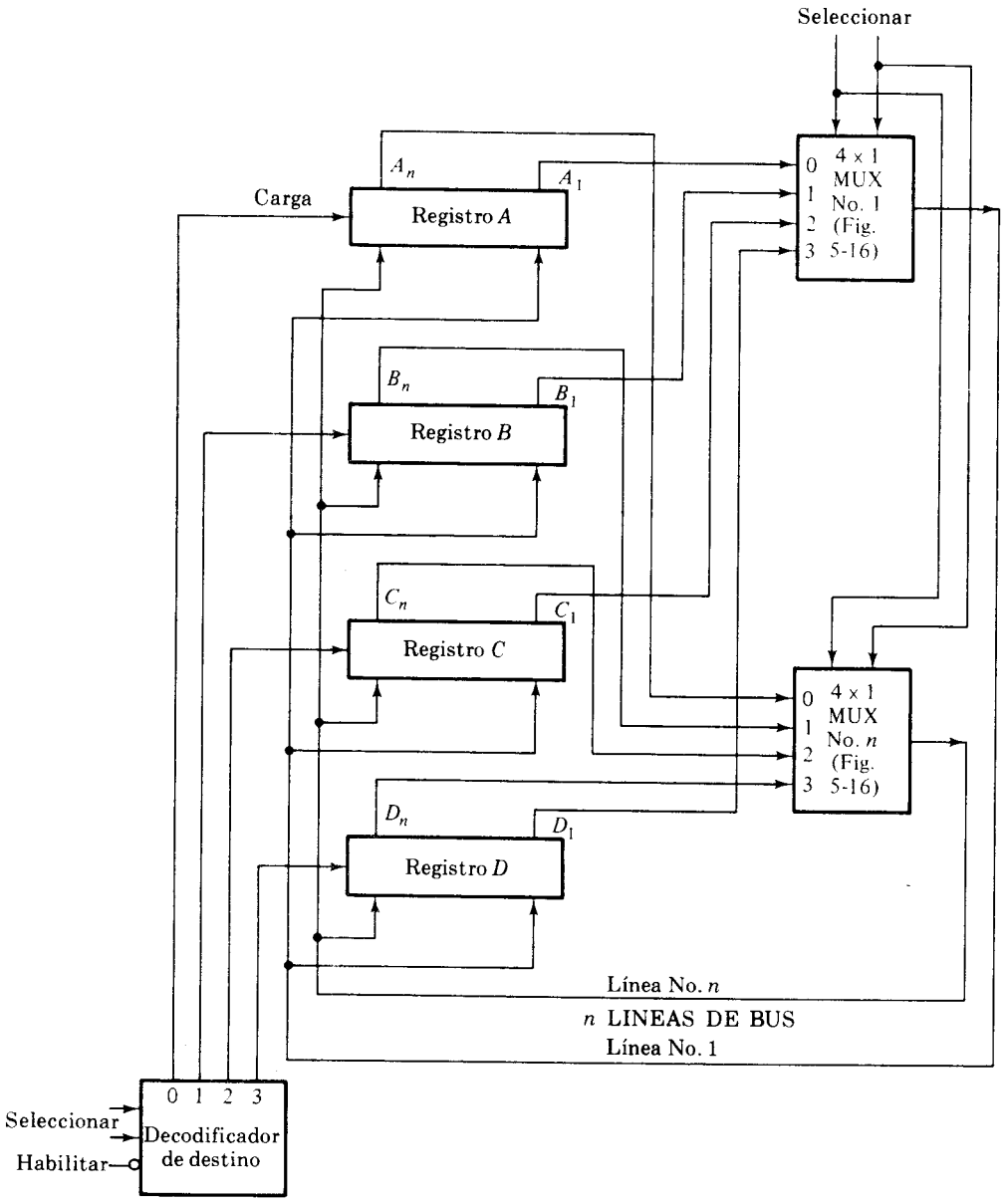


Figura 8-6 Sistema de bus para cuatro registros

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

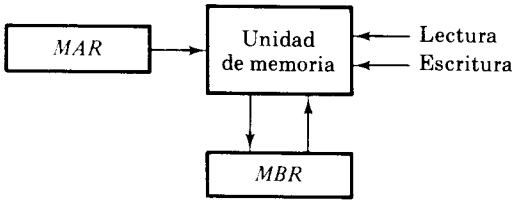


Figura 8-7 Unidad de memoria que se comunica con dos registros externos

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.

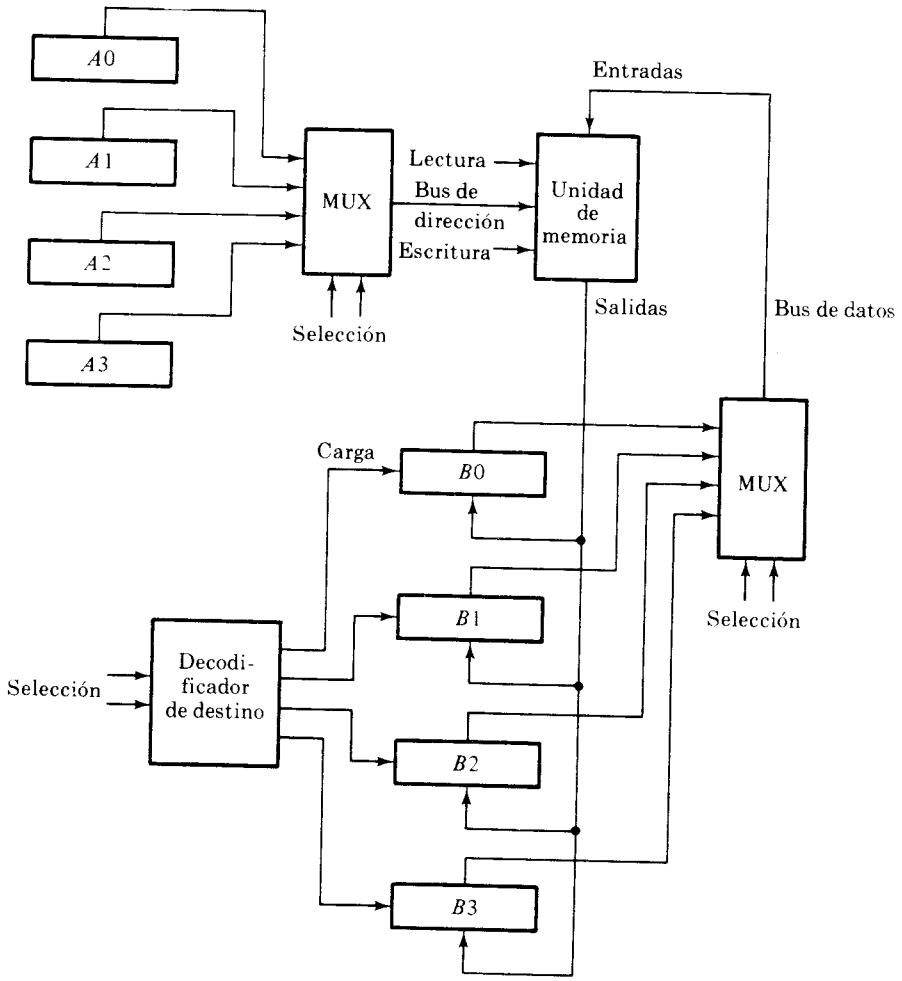


Figura 8-8 Unidad de memoria que se comunica con múltiples registros

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)

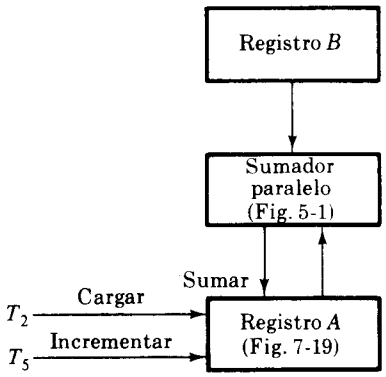


Figura 8-9 Configuración para las microoperaciones de suma e incremento

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
$A \leftarrow \bar{A}$	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:

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

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 más 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

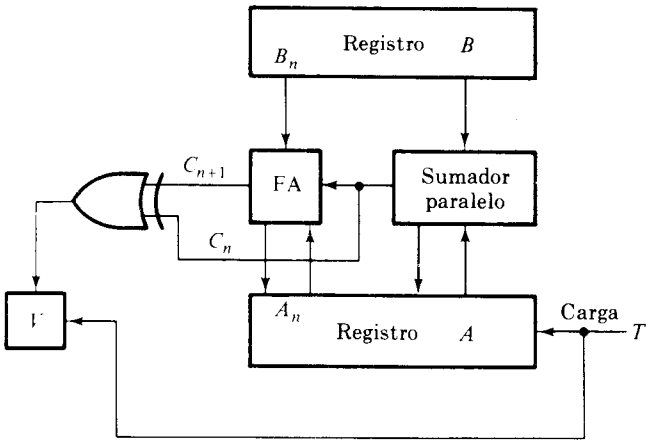


Figura 8-10 Suma de números en signo-complemento de 2

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.

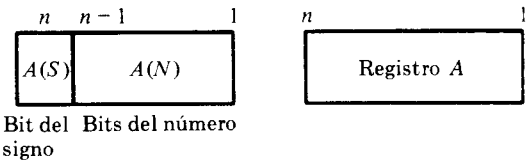


Figura 8-11 Registro que define A para desplazamientos aritméticos

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:

$A(N) \leftarrow \text{shr } A(N), \quad A_{n-1} \leftarrow 0$ para signo-magnitud

$A \leftarrow \text{shr } A, \quad A(S) \leftarrow A(S)$ para signo-complemento de 1 o signo-complemento 2

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:

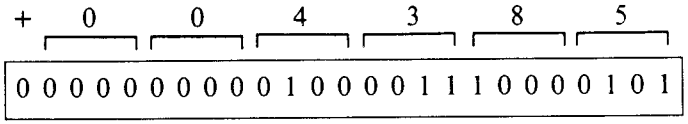
En el caso de signo magnitud, se desplaza y desaparece un 1 de la posición más significativa. En el caso de signo-complemento, ocurrirá la sobrecapacidad si el bit de signo $A_n = A(S)$, no es igual al bit más significativo. Considérese el siguiente ejemplo numérico con números de signo-complemento de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

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

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:

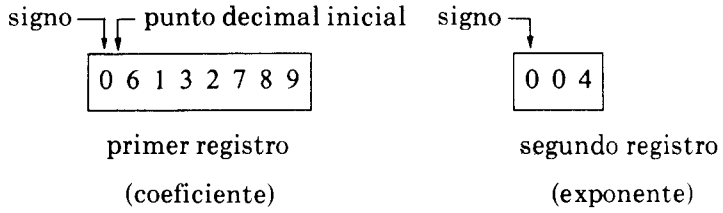
$$A \leftarrow A + \overline{B} + 1$$

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 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.

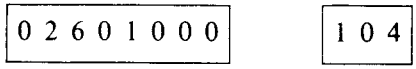
8-9 DATOS DEL PUNTO-FLOTANTE

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:



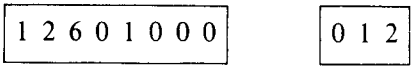
El primer registro tiene un 0 en la posición del flip-flop más significativo para denotar un más. La magnitud del número se almacena en un código binario de 28 flip-flops, con cada dígito decimal ocupando 4 flip-flops. El número en el primer registro se considera una fracción, de manera que el punto 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:



coeficiente exponente

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



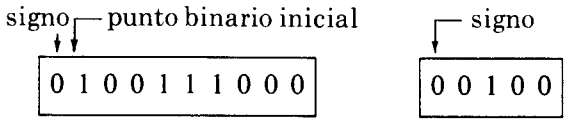
coeficiente exponente

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:



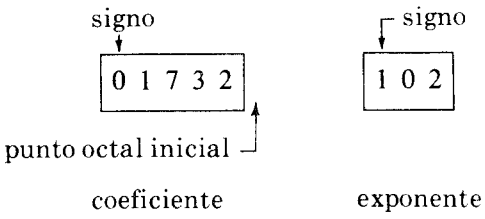
coeficiente exponente

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).

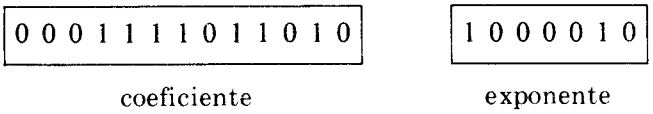
El punto decimal se interpreta en la representación de un número de la siguiente manera:

$$c \cdot r^e$$

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:

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

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:

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

y luego se agrega el nuevo valor:

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

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:

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

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-

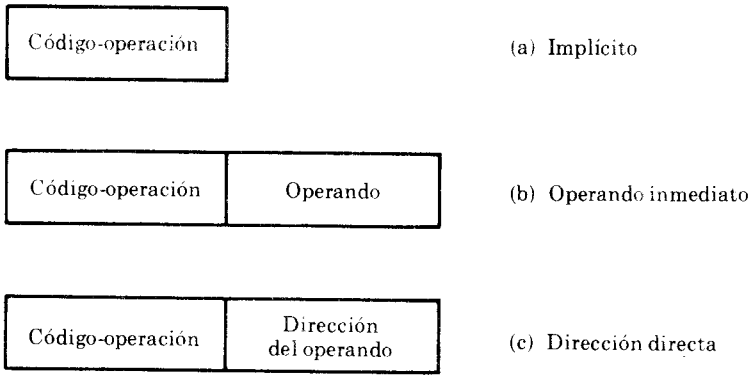


Figura 8-12 Tres formatos posibles de instrucción

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

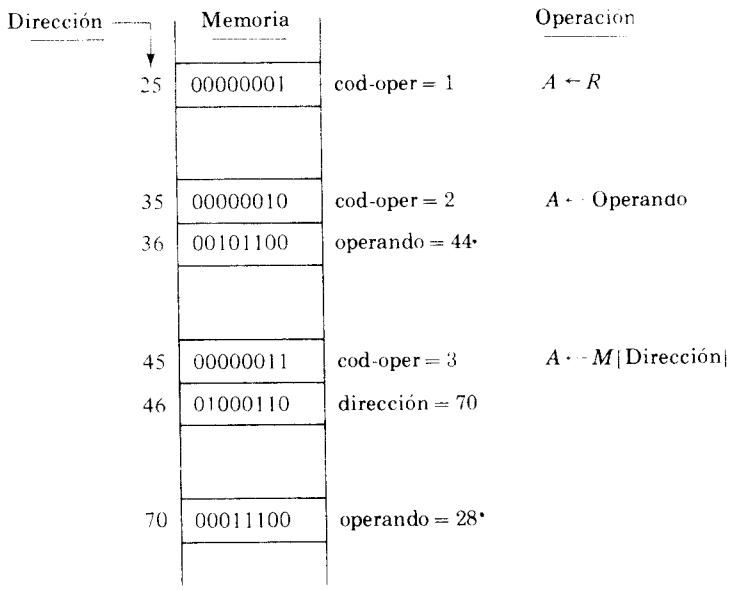


Figura 8-13 Representación de la memoria de instrucciones

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-

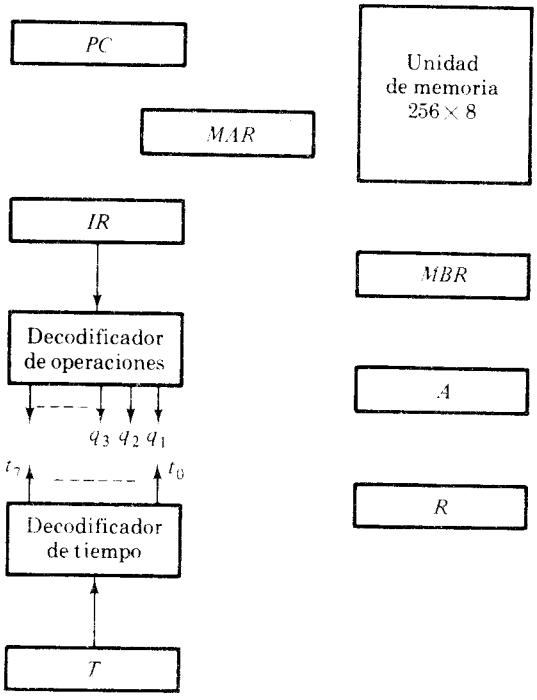


Figura 8-14 Diagrama de bloque de un computador simple

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:

$$q_1 t_3: \quad A \leftarrow R, T \leftarrow 0$$

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:

$q_2 t_3$:	$MAR \leftarrow PC$	trasferir dirección del operando
$q_2 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$	leer el operando, incrementar PC
$q_2 t_5$:	$A \leftarrow MBR, T \leftarrow 0$	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:

$q_3 t_3$:	$MAR \leftarrow PC$	trasferir la siguiente dirección de instrucción
$q_3 t_4$:	$MBR \leftarrow M, PC \leftarrow PC + 1$	leer DIRECCION, (ADRS) incrementar PC
$q_3 t_5$:	$MAR \leftarrow MBR$	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:

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

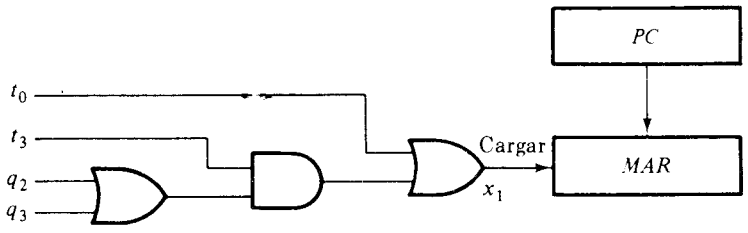


Figura 8-15 Configuración de x_1: $MAR \leftarrow PC$

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

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

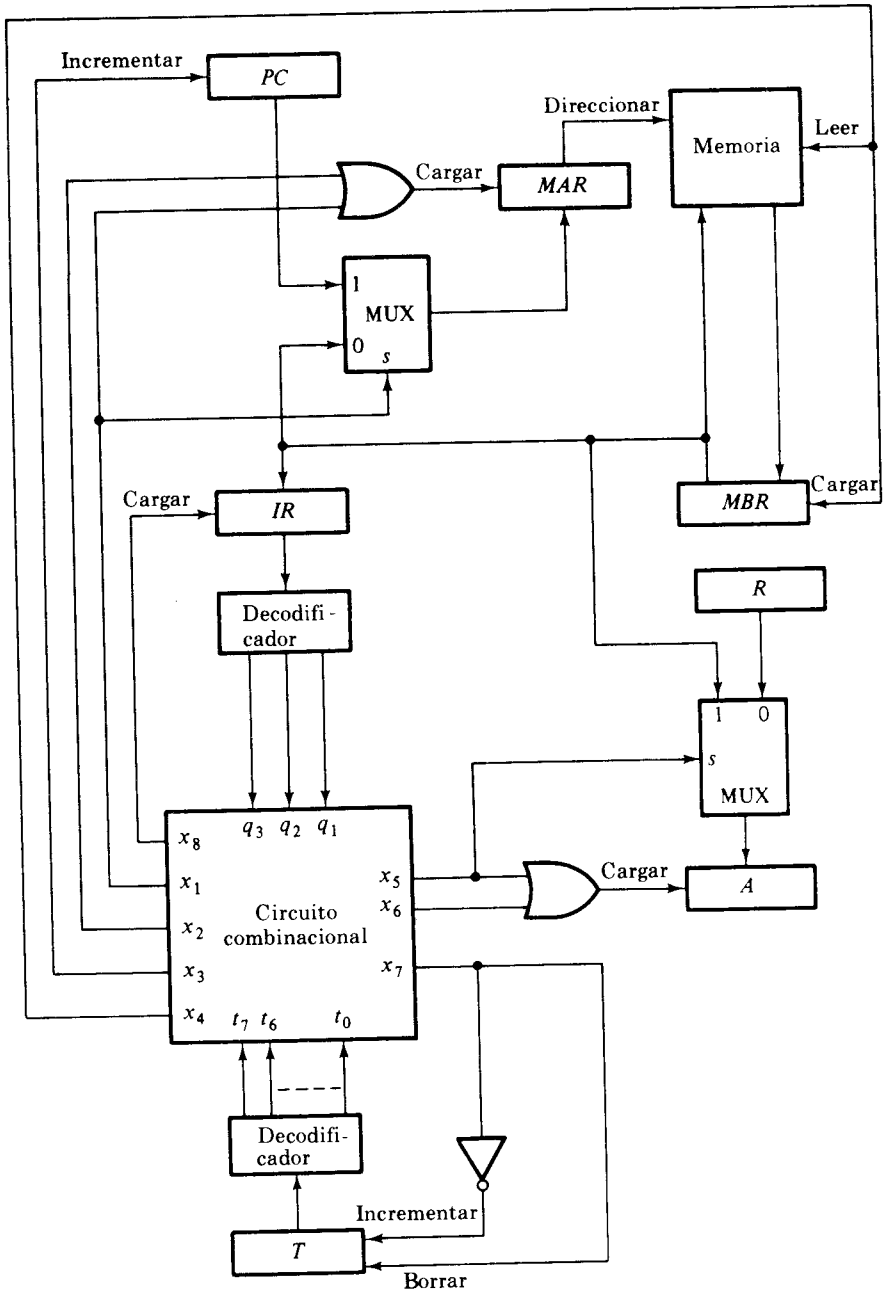


Figura 8-16 Diseño de un computador sencillo

puede construirse con compuertas SSI. En un computador de gran tamaño, esta parte se configura eficientemente con un arreglo lógico programable (PLU).

REFERENCIAS

1. Mano, M. M., *Computer System Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976.
2. Chu, Y., *Computer Organization and Microprogramming.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1972.
3. Dietmeyer, D., *Logical Design of Digital Systems.* Boston, Mass.: Allyn y Bacon, 1971.
4. Bell, C. G y A. Newell, *Computer Structures: Readings and Examples.* Nueva York: McGraw-Hill Book Co., 1971.
5. Hill, F. y G. Peterson, *Digital Systems: Hardware Organization and Design.* Nueva York: John Wiley & Sons, 1973.
6. Bartee, T. C., I. L. Lebow y I. S. Reed, *Theory and Design of Digital Machines.* Nueva York: McGraw-Hill Book Co., 1962.
7. *Computer,* Special Issue on Computer Hardware Description Languages, Vol. 7, No. 12 (diciembre, 1974).
8. *Computer,* Special Issue on Hardware Description Language Applications, Vol. 10, No. 16 (junio, 1977).

PROBLEMAS

8-1. Muestre el diagrama de bloque que ejecuta la proposición:

$$xT_3: \quad A \leftarrow B, \quad B \leftarrow A$$

8-2. Un valor constante puede ser trasferido a un registro aplicando a cada entrada una señal binaria equivalente a lógica 1 o lógica 0. Muestre la configuración de la trasferencia:

$$T: \quad A \leftarrow 11010110$$

8-3. Un registro de 8 bits tiene una entrada x. La operación del registro se describe simbólicamente como:

$$P: \quad A_8 \leftarrow x, \quad A_i \leftarrow A_{i+1} \qquad i = 1, 2, 3, \ldots, 7$$

¿Cuál es la función del registro? Las celdas se numeran de la derecha a la izquierda.

8-4. Muestre la configuración de los materiales (elementos) de las siguientes declaraciones. Los registros tienen 4 bits de longitud.

$$T_0: \quad A \leftarrow R0$$
$$T_1: \quad A \leftarrow R1$$

T_2: $A \leftarrow R2$
T_3: $A \leftarrow R3$

8-5. Sean s_1, s_0 las variables de selección para el multiplexor de la Figura 8-8 y sean $d_1 d_0$ las variables de selección para el decodificador de destino. La variable e se usa para habilitar el decodificador.
(a) Establezca las trasferencias que ocurren cuando las variables de selección $s_1 s_0 d_1 d_0 e$ son iguales a: (1) 00010; (2) 01000; (3) 11100; (4) 01101.
(b) Dé los valores de las variables de selección de las siguientes trasferencias: (1) $A \leftarrow B$; (2) $B \leftarrow C$; (3) $D \leftarrow A$.

8-6. Una unidad de memoria tiene dos entradas de control marcadas como *habilitar* y *lectura/escritura* (como se explica conjuntamente con la Figura 7-30). Las entradas de datos de memoria se conectan a un registro *MBR* como en la Figura 8-7. El *MBR* puede recibir información de un registro externo *EXR* o de la unidad de memoria después de una operación de lectura. El *MBR* suministra los datos para la operación de escritura en memoria. Dibuje un diagrama de bloque usando multiplexores y compuertas que muestren la conexión del *MBR* o la memoria. El sistema debe tener capacidad para ejecutar las siguientes tres trasferencias:

W: $M \leftarrow MBR$	escribir a la memoria
R: $MBR \leftarrow M$	leer de la memoria
E: $MBR \leftarrow EXR$	cargar MBR a partir de EXR

8-7. Las siguientes trasferencias de memoria se especifican para el sistema de la Figura 8-8.

(a) $M[A2] \leftarrow B3$
(b) $B2 \leftarrow M[A3]$
Especifique la operación de memoria y determine las variables de selección binarias para los dos multiplexores y el decodificador de destino.

8-8. Usando los multiplexores cuádruples de 2 a 1 línea de la Figura 5-17 y cuatro inversores, dibuje un diagrama de bloque para configurar las proposiciones:

T_1: $R2 \leftarrow R1$
T_2: $R2 \leftarrow \overline{R2}$
T_3: $R2 \leftarrow 0$

8-9. Considere un registro A de 4 bits con el bit A_4 en la posición más significativa. ¿Cuál es la operación especificada por la siguiente declaración?:

$A_4'C$: $A \leftarrow A + 1$
A_4: $A \leftarrow 0$

Muestre la configuración del sistema usando un contador con carga en paralelo.

8-10. Muestre los componentes necesarios para configurar las siguientes microoperaciones lógicas:

(a) T_1: $F \leftarrow A \wedge B$
(b) T_2: $G \leftarrow C \vee D$
(c) T_3: $E \leftarrow \bar{E}$

8-11. ¿Cuál es la diferencia entre estas dos proposiciones?

$$A + B: \quad F \leftarrow C \vee D$$

y

$$C + D: \quad F \leftarrow A + B$$

8-12. Especifique la trasferencia en serie dibujada en la Figura 7-8 en forma simbólica. Sea S la función de control de desplazamiento. Asuma que S se habilita por un período de cuatro pulsos.

8-13. Muestre los elementos y materiales que configuran la siguiente proposición. Incluya las compuertas lógicas para la función de control.

$$xy'T_0 + T_1 + x'yT_2: \quad A \leftarrow A + B$$

8-14. Un sistema digital tiene tres registros: AR, BR y PR. Los tres flip-flops suministran las funciones de control del sistema. S es un flip-flop el cual es habilitado por una señal externa para comenzar la operación del sistema; F y R se usan para dar secuencia a las microoperaciones. Un cuarto flip-flop, D, se pone a 1 por el sistema digital una vez se complete la operación. La función del sistema se describe por medio de las siguientes operaciones de trasferencia entre registros:

S: $PR \leftarrow 0, \quad S \leftarrow 0, \quad D \leftarrow 0, \quad F \leftarrow 1$

$F \leftarrow 0$, si $(AR = 0)$ entonces $(D \leftarrow 1)$ por tanto $(R \leftarrow 1)$

R: $PR \leftarrow PR + BR, \quad AR \leftarrow AR - 1, \quad R \leftarrow 0, \quad F \leftarrow 1$

¿Cuál es la función que ejecuta el sistema?

8-15. Ejecute las operaciones aritméticas $(+42) + (-13)$ y $(-42) - (-13)$ en binario usando:

(a) Representación en signo-complemento de 1.
(b) Representación en signo-complemento de 2.

8-16. Los números binarios listados a continuación tienen un bit de signo en la posición de extrema izquierda y si son negativos se representan en complemento de 2. Realice las operaciones aritméticas indicadas, usando los algoritmos de suma y resta enunciados en el texto. Compruebe sus resultados haciendo la aritmética con números decimales equivalentes.

(a) 001110 + 110010
(b) 010101 + 000011
(c) 111001 + 001010
(d) 101011 + 111000
(e) 010101 − 000111
(f) 001010 − 111001
(g) 111001 − 001010
(h) 101011 − 100110

8-17. ¿Cuál es el rango de los números que pueden ser acomodados en un registro de 16 bits cuando los números binarios se representan en:

(a) Signo-magnitud?

(b) Signo-complemento de 2?

Dé las respuestas en representación decimal equivalente.

8-18. Ejecute las operaciones aritméticas listadas a continuación con números binarios en representación de signo-complemento de 2 y aplicando el algoritmo enunciado en el texto. Use ocho bits para acomodar cada número conjuntamente con su signo:

(1) $(+65) + (+78)$
(2) $(-65) + (-78)$
(3) $(+35) + (+40)$
(4) $(+65) + (-78)$
(5) $(-65) + (+78)$
(6) $(-35) + (-40)$

Inspeccione la respuesta de 8 bits en cada caso y:

(a) Determine si hay una sobrecapacidad.

(b) Liste los arrastres (carries) que entran o salen de la posición correspondiente al bit de signo.

(c) Determine el signo del resultado (el octavo bit).

(d) Enuncie la relación entre (a) y (b).

(e) Enuncie la relación entre (a) y (c).

8-19. (a) Muestre que el contenido de un registro de 8 bits que almacena los números $+36$ y -36 en binario y en tres representaciones diferentes, es decir, signo-magnitud, signo-complemento de 1 y signo-complemento de 2.

(b) Muestre el contenido del registro después de que los números se desplacen aritméticamente una posición a la derecha (en todas las tres representaciones).

(c) Repita (b) para un desplazamiento a la izquierda.

8-20. Dos números en representación de signo-complemento de 2 se suman de la manera mostrada en la Figura 8-10 y la suma se trasfiere al registro A. Muestre que el desplazamiento aritmético a la derecha simbolizado por:

$$A \leftarrow \text{shr } A, \qquad A_n \leftarrow A_n \oplus V$$

producirá siempre la suma correcta dividida por 2 hubiese o no ocurrido una sobrecapacidad en la suma original.

8-21. Represente $+149$ y -178 en BDC usando la representación de signo-complemento de 10. Use un bit para el signo. Sume los dos números BDC, incluyendo el bit de signo e interprete la respuesta obtenida.

8-22. Los registros para sumar y restar números decimales representados en signo-complemento de 10 es similar a los algoritmos para los números binarios representados en signo-complemento de 2.

(a) Enuncie los algoritmos para la adición y sustracción con representación en signo-complemento de 10. Un signo positivo se representa por un 0 y un signo negativo por un 9 en la posición más significativa.

(b) Aplique los algoritmos para los conjuntos decimales $(-638)+(785)$ y $(-638)-(185)$.

8-23. Un número binario de punto flotante de 36 bits tiene 8 bits más el signo para el exponente. El coeficiente se asume como una fracción normalizada. Los números en el coeficiente y exponente están en la forma de signo-magnitud. ¿Cuáles son las mayores y menores cantidades positivas que pueden ser acomodadas, excluyendo el cero?

8-24. Un registro de 30 bits almacena un número decimal de punto flotante representado en BDC. Los coeficientes ocupan 21 bits del registro y se asume como un entero normalizado. Los números en el coeficiente y exponente se asumen representados en forma de signo-magnitud. ¿Cuáles son las cantidades mayores y menores que pueden ser acomodadas excluyendo el cero?

8-25. Represente el número $(+31{,}5)_{10}$ con un coeficiente entero normalizado de 13 bits y un exponente de 7 bits como:

(a) Un número binario (asuma base de 2).

(b) Un número octal binario codificado (asuma base de 8).

(c) Un número hexadecimal binario codificado (asuma base de 16).

8-26. El registro A almacena la información binaria 11011001. Determine el operando B y la microoperación lógica que se va a realizar entre A y B para cambiar el valor de A a:

(a) 01101101

(b) 11111101

8-27. Determine la operación lógica que borrará selectivamente los bits del registro A en aquellas posiciones donde hay 1 en los correspondientes bits del registro B.

8-28. Un computador digital tiene una unidad de memoria con 24 bits por palabra. El conjunto de instrucciones consiste de 190 operaciones diferentes. Cada instrucción se almacena en una palabra de la memoria y consiste de una parte de código de operación y una parte de dirección.

(a) ¿Cuántos bits se necesitan para el código de operación?

(b) ¿Cuántos bits se dejan para la parte de dirección de la instrucción?

(c) ¿Cuántas palabras pueden acomodarse en la unidad de memoria?

(d) ¿Cuál es el mayor número binario de punto fijo con signo que puede ser almacenado en una palabra de memoria?

8-29. Especifique un formato de instrucción para un computador que realice la siguiente operación:

$$A \leftarrow M[\text{dirección}] + R$$

donde R puede ser cualquiera de los ocho registros posibles en el procesador.

8-30. Asuma que la unidad de memoria de la Figura 8-14 tiene 65,536 palabras de 8 bits cada una.

(a) ¿Cuál debería ser el número de bits de los cinco primeros registros listados en la Tabla 8-4?

(b) ¿Cuántas palabras de memoria se requieren para almacenar la instrucción:

LDA ADRS

como se especifica en la Tabla 8-5?

(c) Liste la secuencia de microoperaciones necesarias para ejecutar la instrucción. El registro R puede ser usado para almacenar temporalmente parte de una dirección.

8-31. Una instrucción inmediata para un simple computador definida en la Figura 8-14 tiene un código de operación 00000100. La instrucción se especifica de la siguiente manera:

LRI OPRD (Cargar OPRD a R) $R \leftarrow OPRD$

Liste la secuencia de microoperaciones para ejecutar esta instrucción.

8-32. Repita el diseño del computador sencillo presentado en la Figura 8-12. Remplace las instrucciones en la Tabla 8-5 por las siguientes instrucciones:

Código de operación	Mnemónico		Descripción	Función
00000001	ADD	R	Sumar directo a A	$A \leftarrow A + R$
00000010	ADI	OPRD	Sumar el operando a A	$A \leftarrow A$ + OPRD
00000011	ADA	ADRS	Sumar R a A	$A \leftarrow A + M$[ADRS]

8-33. Dibuje un diagrama de bloque mostrando la configuración de componentes del sistema especificado en el Problema 8-14. Incluya una entrada de *comienzo* para poner a 1 el flip-flop S y una salida de *hecho* (done) para el flip-flop D.

Diseño lógico de procesadores

9

9-1 INTRODUCCION

Una unidad procesadora es aquella parte de un sistema digital o un computador digital que configura las operaciones en el sistema. Está compuesta por un número de registros y de funciones digitales que conforman microoperaciones aritméticas, lógicas, de desplazamiento y trasferencia. La unidad de proceso se llama una *unidad central de proceso* o CPU, cuando se combina con una unidad de control que supervisa la secuencia de microoperaciones. Este capítulo versa sobre la organización y diseño de la unidad del procesador. El siguiente capítulo trata de la lógica de diseño de la unidad de control. En el Capítulo 11 se demostrará la organización y diseño de un computador CPU.

El número de registros de una unidad procesadora varía desde un registro procesador hasta 64 registros o más. Algunos computadores antiguos vienen con un registro procesador solamente. En algunos casos un sistema digital puede emplear un registro procesador sencillo para propósitos especiales. Sin embargo, como los registros y otras funciones digitales son de bajo costo cuando se construyen con circuitos integrados, todos los computadores recientes emplean un gran número de registros procesadores y canalizan la información entre ellos a través de buses comunes.

Una operación puede ser configurada en una unidad de proceso con una microoperación sencilla o con una secuencia de microoperaciones. Por ejemplo la multiplicación de dos números binarios almacenados en dos registros puede ser configurada con un circuito combinacional que realiza la operación por medio de compuertas. Tan pronto como las señales se propagan a través de las compuertas, el producto estará disponible y puede ser trasferido a un registro de destino con un pulso de reloj sencillo. Alternativamente, la operación de multiplicación puede realizarse con una secuencia de microoperaciones de suma y desplazamiento. El método escogido para la configuración determina la cantidad y tipo de componentes de la unidad de proceso.

Todos los computadores, excepto los muy grandes y rápidos, configuran las operaciones participantes por medio de una secuencia de microoperaciones. De esta manera, el procesador necesita tener solamente circuitos que configuren las microoperaciones básicas simples tales como sumar y desplazar. Otras operaciones, tales como multiplicación, división y aritmetica de punto flotante, se generan conjuntamente con la unidad de control. La unidad procesadora en sí se diseña para configurar microoperaciones básicas del tipo discutido en el Capítulo 8. La unidad de control se diseña para dar secuencia a las microoperaciones que no se incluyen en el conjunto básico.

La función digital que configura las microoperaciones con la información almacenada en los registros del procesador se llama comúnmente *unidad básica aritmética* o ALU. Para realizar una microoperación, el control canaliza la fuente de información de los registros hasta las entradas del ALU. El ALU recibe la información de los registros y realiza una operación dada de la manera especificada por el control. El resultado de la operación se trasfiere al registro de destino. Por definición, el ALU es un circuito combinacional; de manera que toda la operación de trasferencia entre registros pueden realizarse durante el intervalo de un pulso de reloj. Todas las operaciones de trasferencias entre registros, incluyendo la trasferencia entre registros de una unidad procesadora típica, se realizan en un ALU común; de lo contrario, sería necesario duplicar las funciones digitales para cada registro. Las microoperaciones de desplazamiento se realizan a menudo en una unidad separada. Una unidad de desplazamiento se muestra por lo general separada, pero algunas veces está incluida como parte de la unidad enteramente aritmética y lógica.

Un computador CPU debe manipular no solamente datos sino también códigos de instrucción y direcciones que vienen de la memoria. El registro que almacena y manipula el código de operación de instrucciones se considera como parte de la unidad de control. Los registros que almacenan direcciones son incluidos algunas veces como parte de la unidad de proceso y la información de direcciones se procesa por un ALU común. En algunos computadores, los registros que almacenan direcciones son conectados a un bus separado y la información de dirección se manipula con funciones digitales separadas.

Este capítulo presenta varias alternativas para la organización y diseño de una unidad de proceso. El diseño de una unidad aritmética lógica particular se lleva a cabo para mostrar el proceso de diseño usado en la formulación e implementación de una función digital común capaz de realizar un gran número de microoperaciones. Otras funciones digitales consideradas y diseñadas en este capítulo son la unidad de desplazamiento y el registro procesador para propósitos generales, comúnmente llamado *acumulador*.

9-2 ORGANIZACION DEL PROCESADOR

La parte procesadora de un computador CPU se trata algunas veces como *el canal de datos* del CPU porque el procesador formula los canales de trasferencia de datos entre los registros de la unidad. Los diferentes caminos

son controlados supuestamente por medio de compuertas que abren los caminos necesarios y cierran otros. Una unidad procesadora puede diseñarse para satisfacer un conjunto de canales de datos para una aplicación específica. El diseño de un procesador para propósitos especiales fue demostrado en la Sección 8-9. La Figura 8-16 muestra los diferentes canales de datos para un procesador muy limitado. La abertura de los canales o caminos de datos se logra por medio de decodificadores y circuitos combinacionales que comprenden la sección de control de la unidad.

En una unidad procesadora bien organizada, los canales de datos se forman por medio de buses y otras líneas comunes. Las compuertas de control que formulan los canales de datos son esencialmente multiplexores y decodificadores cuyas líneas de selección especifican el camino requerido. El proceso de información se hace mediante una función digital común cuyo canal de datos puede ser especificado por un conjunto de variables de selección comunes. Una unidad procesadora que tiene una organización bien estructurada puede usarse en una gran cantidad de operaciones. Si se construye dentro de un circuito integrado, se hará disponible para muchos usuarios ya que para cada uno se puede tener una aplicación diferente.

En esta sección, se investigan varias alternativas para organizar una unidad procesadora para propósitos generales. Todas las organizaciones emplean un ALU común y un registro de desplazamiento. Las diferencias en las organizaciones se manifiestan principalmente en la organización de los registros y sus canales comunes al ALU.

Organización del bus

Cuando se incluye un gran número de registros en una unidad de proceso es más eficiente conectarlos por medio de buses comunes o arreglarlos como una memoria pequeña que tiene un tiempo de acceso muy rápido. Los registros se comunican entre sí no solamente por la trasferencia directa de datos sino también cuando se realizan varias microoperaciones. En la Figura 9-1 se muestra una organización con bus para cuatro registros procesadores. Cada registro se conecta a dos multiplexores (MUX) para formar los buses de entrada A y B. Las líneas de selección de cada multiplexor seleccionan un registro para el bus particular. Los buses A y B se aplican a una unidad lógica aritmética común. La función seleccionada en el ALU determina la operación particular que se va a realizar. Las microoperaciones de desplazamiento se configuran en el registro de desplazamiento. El resultado de la microoperación pasa a través del bus de salida S hasta las entradas de todos los registros. El registro de destino que recibe la información del bus de salida se selecciona por medio de un decodificador. Cuando se habilita, este decodificador activa una de las entradas de carga del registro para suministrar un canal de trasferencia entre los datos del bus S y las entradas del registro de destino seleccionado.

El bus de salida S alimenta los terminales para trasferir datos de un destino externo. Una entrada del multiplexor A o B puede recibir datos de los elementos que lo rodean cuando es necesario trasferir datos externos a la unidad de proceso.

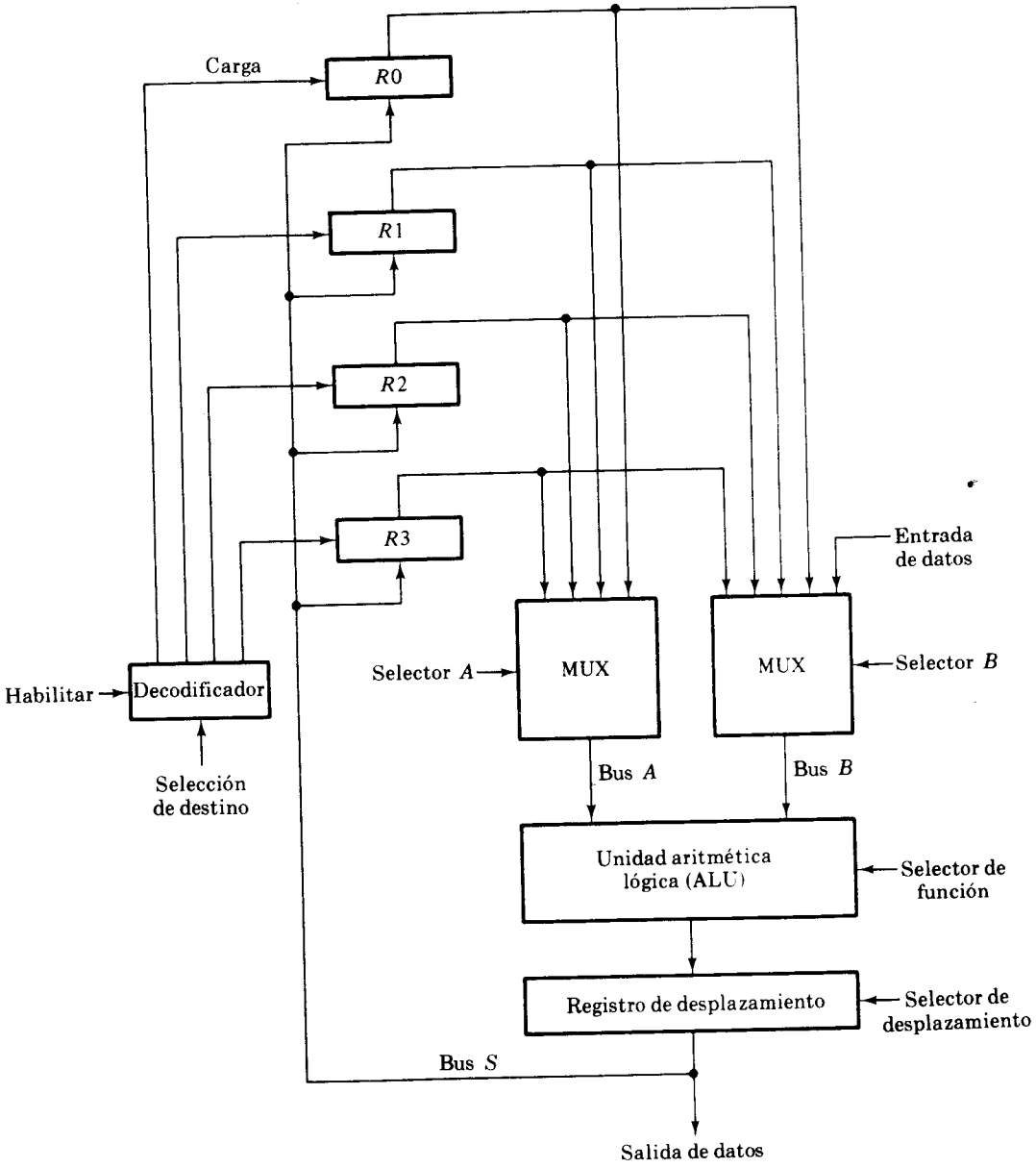


Figura 9-1 Registros procesadores y ALU conectados por medio de buses comunes

La operación de los multiplexores, los buses y el decodificador de destino se explica en la Sección 8-2 conjuntamente con la Figura 8-6. El ALU y el registro de desplazamiento se discuten más tarde en este capítulo.

Una unidad procesadora puede tener más de cuatro registros. La construcción de un procesador con bus organizado con más registros requiere multiplexores mayores y decodificador; de otra forma sería similar a la organización presentada en la Figura 9-1.

La unidad de control que supervisa el sistema de bus procesador dirige el flujo de información a través del ALU seleccionando los diferentes componentes de la unidad. Por ejemplo para realizar la microoperación:

$$R1 \leftarrow R2 + R3$$

el control debe suministrar variables de selección binarias a las siguientes entradas de selección:

1. Selector MUX A: coloca el contenido de $R2$ en el bus A.
2. Selector MUX B: coloca el contenido de $R3$ en el bus B.
3. Selector de función ALU: genera la operación aritmética $A + B$.
4. Selector de desplazamiento: para la trasferencia directa de la salida del ALU al bus de salida S (ningún desplazamiento).
5. Selector de destino del decodificador: trasfiere el contenido del bus S a $R1$.

Las cinco variables selectivas de control deben ser generadas simultáneamente y deben estar disponibles durante un intervalo de pulso de reloj común. La información binaria de los dos registros fuente se propaga a través de las compuertas combinacionales en los multiplexores, el ALU y el registro de desplazamiento hasta el bus de salida y a las entradas del registro de destino durante un intervalo de pulso de reloj, la información binaria en el bus de salida se trasfiere al $R1$ cuando se presenta el siguiente pulso de reloj. Para lograr una rápida respuesta de tiempo, se construye el ALU con circuitos generadores de arrastre posterior y el registro de desplazamiento se configura con compuertas combinacionales.

Cuando se encapsula en un CI, la unidad procesadora se llama algunas veces *registro y unidad lógica aritmética* o RALU (register and arithmetic logic unit). Algunos fabricantes lo llaman *un microprocesador de un grupo de bits*. El prefijo *micro* se refiere a un tamaño físico muy pequeño del circuito integrado en el cual se incluye el procesador. *El grupo de bits* se refiere al hecho de que el procesador puede ser expandido a una unidad de proceso con un gran número de bits usando un grupo de CI. Por ejemplo, un microprocesador de un grupo de 4 bits contiene registros y ALU para manipular datos de 4 bits. Dos CI de éstos pueden ser combinados para construir una unidad procesadora de 8 bits. Para un procesador de 16 bits, es necesario usar cuatro circuitos integrados y conectarlos en cascada. El arrastre de salida de un ALU se conecta al arrastre de entrada del siguiente ALU de mayor orden y la salida en serie y líneas de entrada de los re-

gistros de desplazamiento se conectan también en cascada. *Un microprocesador de un grupo de bits* debe distinguirse de otro tipo de CI llamado *microprocesador*. El primero es una unidad procesadora mientras que el microprocesador se refiere a un computador CPU completo encapsulado en una pastilla de CI. Los microprocesadores y su equipo asociado se discutirán en el Capítulo 12.

Memoria "scratchpad" o memoria tapón

Los registros de una unidad procesadora pueden ser metidos dentro de una unidad pequeña de memoria. Cuando estos se incluyen en la unidad de proceso, la memoria pequeña se llama memoria *tapón* o de borrado. El uso de una pequeña memoria es una alternativa muy económica para conectar los registros procesadores a través del sistema de bus. La diferencia entre dos sistemas es la manera en la cual la información se selecciona para la trasferencia al ALU. En el sistema de bus, la trasferencia de información se selecciona por medio de los multiplexores que forman los buses. Por otra parte, un solo registro dentro de un grupo de registros organizados como una pequeña memoria puede ser seleccionado por medio de una dirección de la unidad de memoria. Un registro de memoria puede funcionar justamente como cualquier otro registro procesador ya que su única función es almacenar información binaria para ser procesada en el ALU.

Una memoria tapón o de borrado debe distinguirse de la memoria principal del computador. En contraste con la memoria principal, la cual almacena instrucciones y datos, una pequeña memoria de una unidad de proceso es meramente una alternativa para conectar un número de registros procesadores por medio de un camino de trasferencia común. La información almacenada en una memoria tapón o de borrado debe venir normalmente de la memoria principal por medio de instrucciones en el programa.

Considérese, por ejemplo, una unidad procesadora que emplea ocho registros de 16 bits cada uno. Los registros pueden incluirse dentro de una memoria pequeña de ocho palabras de 16 bits cada una, o un RAM de 8×16. Las ocho palabras de memoria pueden designarse como $R0$ hasta $R7$, correspondiendo a las direcciones 0 hasta 7 y constituyen los registros para el procesador.

Una unidad procesadora que usa una memoria tapón o de borrado se muestra en la Figura 9-2. Un registro fuente se selecciona de la memoria y se carga al registro A. Un segundo registro fuente se selecciona de la memoria y se carga al registro B. La selección se hace especificando las direcciones de palabra correspondientes y activando la entrada de lectura de la memoria. La información de A y B se manipula en el ALU y en el registro de desplazamiento. El resultado de la operación se trasfiere a un registro de memoria especificando su dirección de palabra y activando el control de entrada de escritura en memoria. El multiplexor a la entrada de la memoria puede seleccionar datos de entrada de una fuente externa.

Asúmase que la memoria tiene ocho palabras, de manera que una dirección puede especificarse con tres bits. Para realizar la operación:

$$R1 \leftarrow R2 + R3$$

el control debe suministrar las variables de selección binarias para realizar la siguiente secuencia de tres microoperaciones:

T_1:	$A \leftarrow M[010]$	leer $R2$ al registro A
T_2:	$B \leftarrow M[011]$	leer $R3$ al registro B
T_3:	$M[001] \leftarrow A + B$	ejecutar una operación en el ALU y trasferir el resultado a $R1$

La función de control T_1 debe suministrar la dirección 010 a la memoria y activar las entradas A de *lectura* y *carga*. La función de control T_2 debe alimentar una dirección 011 a la memoria y activar las entradas B de *lectura* y *carga*. La función de control T_3 debe suministrar el código de función al ALU y al registro de desplazamiento para ejecutar la operación de *suma* (sin desplazamiento), aplicar una dirección 001 a la memoria, seleccionar la salida del registro de desplazamiento para el MUX y activar la entrada de *escritura* de memoria. El símbolo $M[xxx]$ designa una palabra de memoria (o registro) especificada por una dirección dada en el número binario xxx.

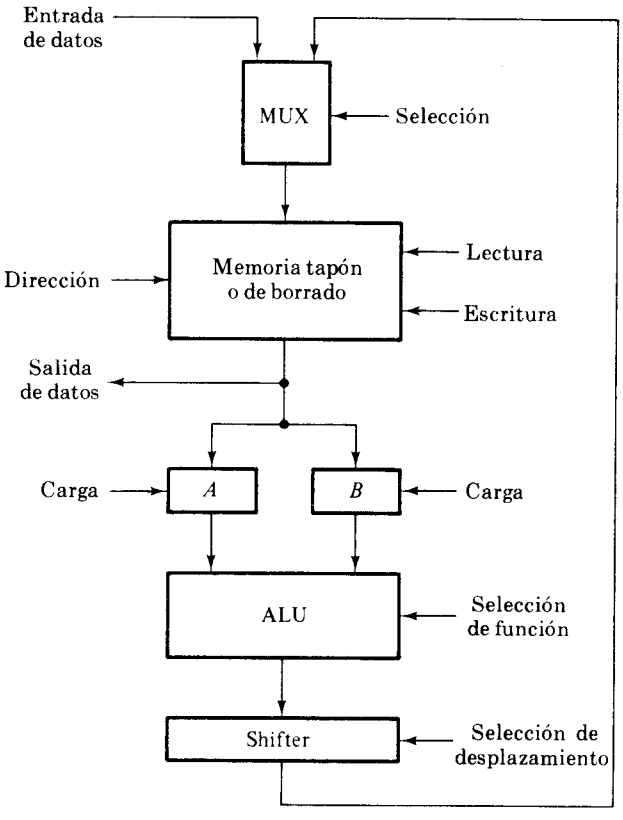


Figura 9-2 Unidad de proceso que emplea una memoria tapón

La razón de una secuencia de tres microoperaciones en vez de una, como en un procesador con organización de bus, se debe a la limitación de la unidad de memoria. Como la unidad de memoria tiene solamente un grupo de terminales de dirección y se va a comunicar con dos registros fuente, se necesitan dos vías de acceso a la memoria para leer la información de la fuente. La tercera microoperación es necesaria para direccionar el registro de destino. Si el registro de destino es el mismo que el segundo registro fuente, el control podría activar la entrada de lectura, para extraer la información de la segunda fuente, seguida de una señal de escritura para activar la trasferencia de destino y sin tener que cambiar el valor de la dirección.

Algunos procesadores emplean una memoria de 2 puertos para poder vencer la demora causada al leer dos registros fuentes. Una memoria de 2 puertos tiene dos líneas de dirección separadas para seleccionar las palabras de memoria simultáneamente. De esta manera pueden leerse los dos registros fuente al mismo tiempo. Si el registro de destino es igual a uno de los registros fuente, entonces toda la microoperación puede hacerse durante el período de un pulso de reloj.

La organización de una unidad procesadora con una memoria de 2 puertos se muestra en la Figura 9-3.* La memoria tiene dos grupos de direcciones, una para el puerto A y otra para el puerto B. Los datos de cualquier palabra en la memoria se leen en registro A especificando una dirección A. De igual manera cualquier palabra de memoria se lee al registro B especificando una dirección B. La misma dirección puede ser aplicada a la

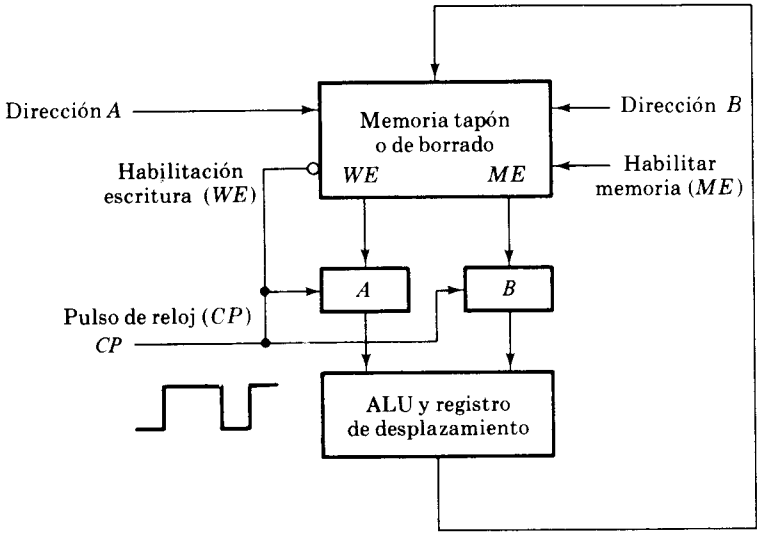


Figura 9-3 Unidad de proceso con una memoria de 2 puertos

*Esta organización es similar al microprocesador de un grupo de bits, tipo 2901.

dirección A y a la dirección B, en cuyo caso aparecerá una palabra idéntica en ambos registros A y B. Cuando se habilitan por medio del terminal habilitador de memoria (ME = memory enable), se pueden escribir nuevos datos a la palabra especificada por la dirección B. Así las direcciones de A y B especifican dos registros fuente simultáneamente y la dirección B especifica siempre el registro de destino. La Figura 9-3 no muestra un camino para datos externos de entrada y salida, pero pueden ser incluidos como en las organizaciones anteriores.

Los registros A y B son, en efecto, retenedores que aceptan nueva información siempre y cuando el pulso de reloj CP esté en el estado 1; cuando CP vaya a 0, los retenedores se inhabilitan y retienen la información que estaba almacenada cuando CP era un 1. Esto elimina cualquier condición de congestión que puede ocurrir cuando se está escribiendo la nueva información en la memoria. La entrada del reloj controla las operaciones de lectura y escritura en memoria por medio del terminal de habilitación de escritura (write enable). Este controla las trasferencias a los retenedores A y B. La forma de onda de un intervalo de un pulso de reloj se muestra en el diagrama.

Cuando el terminal de reloj es 1, los retenedores A y B se abren y aceptan la información que viene de la memoria. El terminal WE está también en el estado 1. Este habilita la operación de escritura y de lectura en la memoria. Así cuando $CP = 1$ las palabras seleccionadas por las direcciones A y B se leen de la memoria y se colocan en los registros A y B respectivamente. La operación en el ALU se realiza con los datos almacenados en A y B. Cuando el terminal del reloj va a 0, los retenedores se cierran y se retienen los últimos datos introducidos. Si el terminal de ME está habilitado cuando $WE = 0$, el resultado de la microoperación se escribe en la palabra de memoria definida por la dirección B. Así una microoperación:

$$R1 \leftarrow R1 + R2$$

puede hacerse dentro de un período de un pulso de reloj. El registro de memoria $R1$ debe especificarse con la dirección B y $R2$ con la dirección A.

Registro acumulador

Algunas unidades procesadoras separan un registro de otros y se le llama registro *acumulador,* abreviado AC o registro A. El nombre de este registro se deriva del proceso de adición aritmética que se encuentra en los computadores digitales. El proceso de sumar muchos números se lleva a cabo almacenando inicialmente esos números en otros registros procesadores o en la unidad de memoria del computador y borrando el acumulador a 0. Los números se agregan al acumulador uno a uno en orden consecutivo. El primer número se agrega a 0 y la suma se trasfiere al acumulador. El segundo número se agrega a los contenidos del acumulador y la suma formada de nuevo remplaza su valor previo. Este proceso se continúa hasta que todos los números se agregan y se forma la suma total. Así, el registro "acumula" la suma paso a paso haciendo sumas secuenciales entre un número nuevo y la suma acumulada previamente.

El registro acumulador en una unidad de proceso es un registro multipropósito capaz de realizar no solamente la microoperación de suma sino también otras microoperaciones de la misma forma. De hecho, las compuertas asociadas con un registro acumulador suministran todas las funciones digitales encontradas en un ALU.

La Figura 9-4 muestra el diagrama de bloque de una unidad procesadora que emplea un registro acumulador. El registro A se distingue de todos los demás registros procesadores. En algunos casos toda la unidad procesadora es justamente el registro acumulador y el ALU asociado. El registro en sí puede funcionar como un registro de desplazamiento para suministrar las microoperaciones de desplazamiento. La entrada B suministra una fuente de información externa. Esta información puede provenir de otros registros procesadores o directamente de la memoria principal del computador. El registro A suministra la otra fuente de información al ALU por el terminal A. El resultado de una operación se trasfiere de nuevo al registro A y se remplaza su contenido previo. La salida del registro A puede ir a un destino externo o a los terminales de entrada de otros registros procesadores o unidad de memoria.

Para formar la suma de dos números almacenados en los registros procesadores, es necesario agregarlos en el registro A usando la siguiente secuencia de microinstrucciones:

T_1: $A \leftarrow 0$ borrar A

T_2: $A \leftarrow A + R1$ trasferir $R1$ a A

T_3: $A \leftarrow A + R2$ agregar $R2$ a A

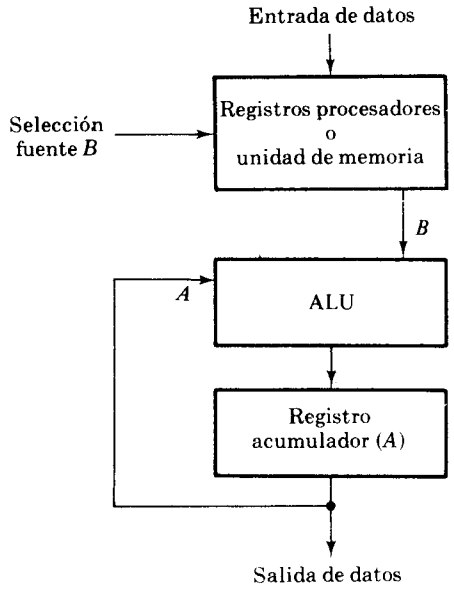


Figura 9-4 Procesador con un registro acumulador

El registro A se borra primero. El primer número en $R1$ se trasfiere al registro A agregando al actual contenido de ceros de A. El segundo número en $R2$ se agrega al valor presente de A. La suma formada en A debe usarse para otros cálculos o puede ser trasferida a su destino requerido.

9-3 UNIDAD LOGICA ARITMETICA

Una unidad lógica aritmética (ALU) es una función multioperación digital de lógica combinacional. Esta puede realizar un conjunto de operaciones aritméticas básicas y un conjunto de operaciones lógicas. El ALU tiene un número de líneas de selección para seleccionar una operación particular de la unidad. Las líneas de selección se decodifican dentro del ALU de manera que las k variables de selección pueden especificar hasta 2^k operaciones diferentes.

La Figura 9-5 muestra el diagrama de bloque de un ALU de 4 bits. Las cuatro entradas de datos de A se combinan con las cuatro entradas de B para generar una operación en las salidas F. El terminal de selección de modo s_2 distingue entre las operaciones aritméticas y lógicas. Las dos entradas de selección de función s_1 y s_0 especifican la operación aritmética o lógica que se va a generar. Con tres variables de selección es posible especificar cuatro operaciones aritméticas (con s_2 en un estado) y cuatro operaciones lógicas (con s_2 en el otro estado). Los arrastres de entrada y salida tienen significado solamente durante una operación aritmética.

El arrastre de entrada en la posición menos significativa de un ALU se usa muy a menudo como una cuenta variable de selección que puede doblar el número de operaciones aritméticas. De esta manera, es posible generar cuatro operaciones más, para un total de ocho operaciones aritméticas.

Un diseño de un ALU típico se llevará a cabo en tres etapas. Primero, será emprendido el diseño de la sección aritmética. Segundo, debe considerarse el diseño de la sección lógica. Finalmente, deberá modificarse la sec-

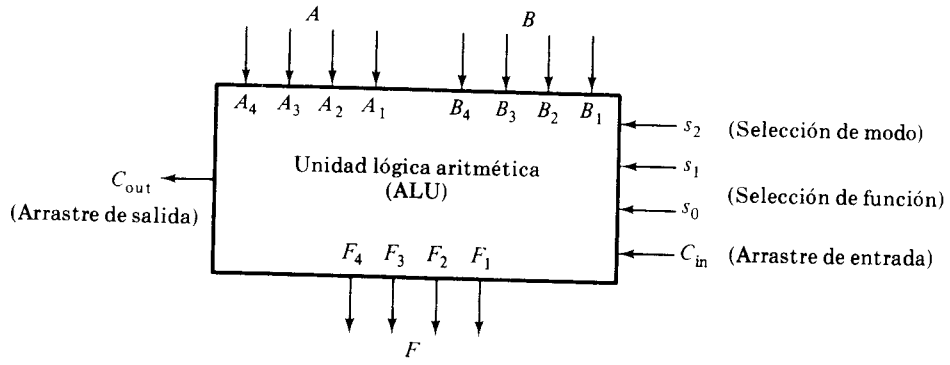


Figura 9-5 Diagrama de bloque de un ALU de 4 bits

ción aritmética de manera que puedan realizarse ambas operaciones aritméticas y lógicas.

9-4 DISEÑO DE UN CIRCUITO ARITMETICO

El componente básico de la sección aritmética de un ALU es un sumador en paralelo. Un sumador en paralelo se construye con un número de cir-

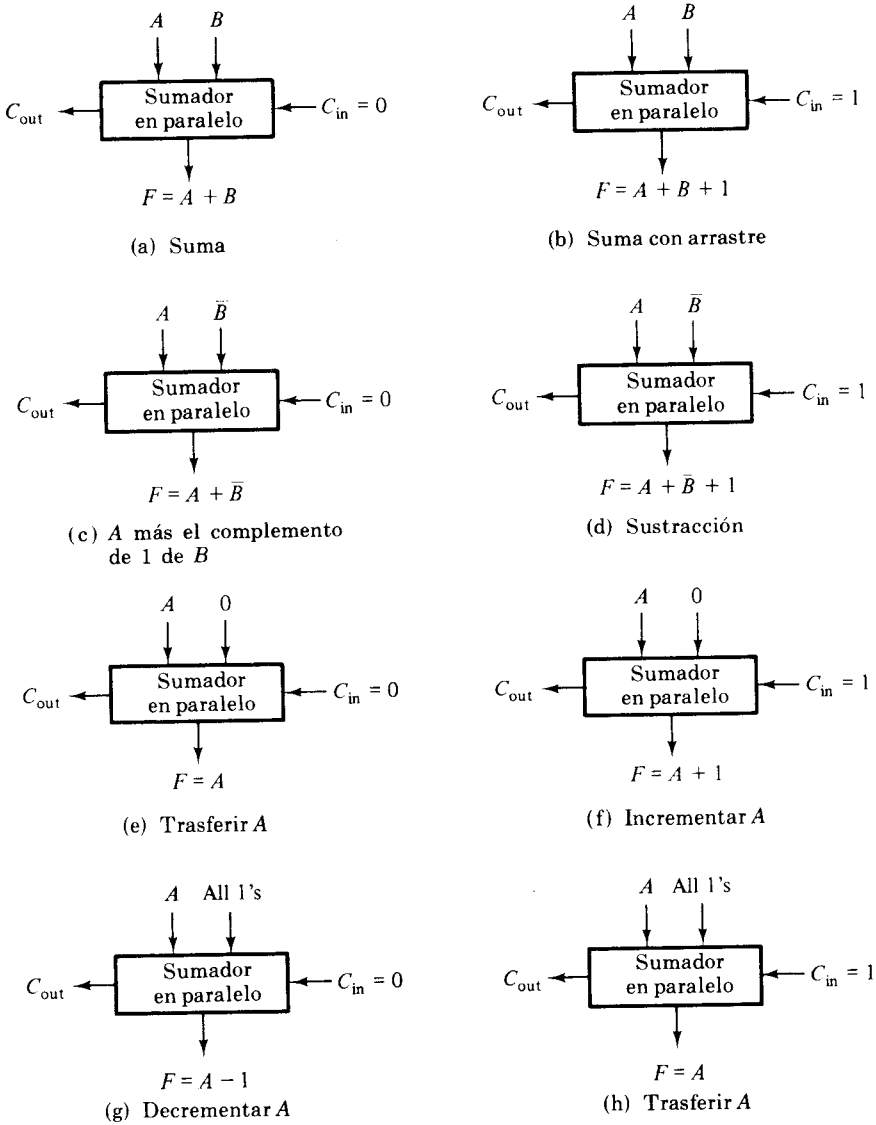


Figura 9-6 Operaciones obtenidas mediante el control de un grupo de entradas de un sumador en paralelo

cuitos sumadores completos conectados en cascada (ver Sección 5-2). Controlando la entrada de datos al sumador en paralelo, es posible obtener diferentes tipos de operaciones aritméticas. La Figura 9-6 muestra las operaciones aritméticas obtenidas cuando un grupo de entradas a un sumador en paralelo se controlan externamente. El número de bits en el sumador en paralelo puede tener cualquier valor. El arrastre de entrada C_{in} pasa al circuito sumador completo a la posición del bit menos significativo. El arrastre de salida C_{out} proviene del circuito sumador completo de la posición del bit más significativo.

La suma aritmética se logra cuando un grupo de entradas recibe un número binario A, el otro conjunto de entradas recibe un número binario B y el arrastre de entrada se mantiene en 0. Esto se muestra en la Figura 9-6(a). Haciendo $C_{in} = 1$ como en la Figura 9-6(b), es posible agregar 1 a la suma en F. Considérese ahora el efecto de completar todos los bits de la entrada B. Cuando $C_{in} = 0$, la salida produce $F = A + \overline{B}$, la cual es la suma de A más el complemento de 1 de B. Agregando 1 a esta suma y haciendo $C_{in} = 1$ se obtiene $F = A + \overline{B} + 1$ lo cual produce la suma A más el complemento de 2 de B. Esta operación es similar a la operación de sustracción si se descarta el arrastre de salida. Si se colocan sólo ceros a los terminales B, se obtiene $F = A + 0 = A$, lo cual trasfiere la entrada A a la salida F. Agregando un 1 a C_{in} como en la Figura 9-6(f), se obtiene $F = A + 1$ lo cual es la operación de incremento.

La condición ilustrada en la Figura 9-6(g) colocará todos los 1 en los terminales B. Esto produce la operación de decremento $F = A - 1$. Para mostrar que esta condición es una operación de decremento, considérese un sumador en paralelo con n circuitos sumadores completos. Cuando el arrastre de salida es 1 éste representa el número 2^n, porque 2^n en binario consiste de un 1 seguido por n ceros. Restando 1 de 2^n, se obtiene $2^n - 1$, lo cual en binario es un número de n unos. Sumando $2^n - 1$ a A se obtiene $F = A + 2^n - 1 = 2^n + A - 1$. Si se suprime el arrastre de salida 2^n se obtiene $F = A - 1$. Para hacer una demostración con un ejemplo numérico, sea $n = 8$ y $A = 9$. Entonces:

$$\begin{aligned} A &= \quad 0000\ 1001 = (9)_{10} \\ 2^n &= 1\ 0000\ 0000 = (256)_{10} \\ 2^n - 1 &= \quad 1111\ 1111 = (255)_{10} \\ A + 2^n - 1 &= 1\ 0000\ 1000 = (256 + 8)_{10} \end{aligned}$$

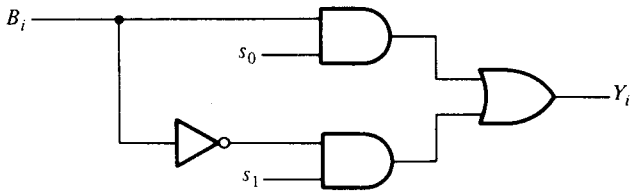
Quitando el arrastre de salida $2^n = 256$, se obtiene $8 = 9 - 1$. Así, se ha decrementado A en 1 agregándole un número binario con sólo unos.

El circuito que controla la entrada B para suministrar las funciones ilustradas en la Figura 9-6 se llaman elemento *verdadero/complemento, uno/cero*. Este circuito se ilustra en la Figura 9-7. Las dos líneas de selección s_1 y s_0 controlan la entrada de cada terminal B. El diagrama muestra una entrada típica designada por B_i y una salida designada por Y_i. En una aplicación típica, hay n circuitos para $i = 1, 2, \ldots, n$. Como se muestra en la tabla de la Figura 9-7, cuando ambos s_0 y s_1, sean iguales a 0, la salida $Y_i = 0$, independientemente del valor de B_i. Cuando $s_1 s_0 = 01$ la

compuerta AND superior genera el valor de B_i mientras que la salida de la compuerta inferior es 0; de manera que $Y_i = B_i$. Cuando $s_1 s_0 = 10$, la compuerta AND inferior genera el complemento de B_i para dar $Y_i = B_i'$. Cuando $s_i s_0 = 11$, ambas compuertas estarán activas y $Y_i = B_i + B_i' = 1$.

Un circuito aritmético de 4 bits que realiza ocho operaciones aritméticas se muestra en la Figura 9-8. Los cuatro circuitos sumadores completos (FA) constituyen el sumador en paralelo. El arrastre que va a la primera etapa es el arrastre de entrada, el arrastre de salida de la cuarta etapa es el arrastre de salida. Todos los demás arrastres están conectados internamente de una etapa a la siguiente. Las variables de selección son s_1, s_0 y C_{in}. Las variables s_1 y s_0 controlan todas las entradas B a los circuitos del sumador completo como en la Figura 9-7. Las entradas A van directamente a las otras entradas de los sumadores completos.

Las operaciones aritméticas configuradas en el circuito aritmético se listan en la Tabla 9-1. Los valores de las entradas AND a los circuitos sumadores completos son una función de las variables de selección s_1 y s_0. Agregando el valor de Y en cada caso al valor de A más el valor de C_{in}, da la operación aritmética en cada entrada. Las ocho operaciones listadas en la tabla se desprenden directamente de los diagramas de función ilustrados en la Figura 9-6.



s_1	s_0	Y_i
0	0	0
0	1	B_i
1	0	B_i'
1	1	1

Figura 9-7 Circuito verdadero/complemento, uno/cero

Tabla 9-1 Tabla de función para el circuito aritmético de la Figura 9-8

Selector de función			Y igual a	Salida igual a	Función
s_1	s_0	C_{in}			
0	0	0	0	$F = A$	Trasferir A
0	0	1	0	$F = A + 1$	Incrementar A
0	1	0	B	$F = A + B$	Agregar B a A
0	1	1	B	$F = A + B + 1$	Agregar B a A más 1
1	0	0	$\overline{B}$	$F = A + \overline{B}$	Agregar el complemento de 1 de B a A
1	0	1	$\overline{B}$	$F = A + \overline{B} + 1$	Agregar el complemento de 2 de B a A
1	1	0	Todo unos	$F = A - 1$	Decrementar A
1	1	1	Todo unos	$F = A$	Trasferir A

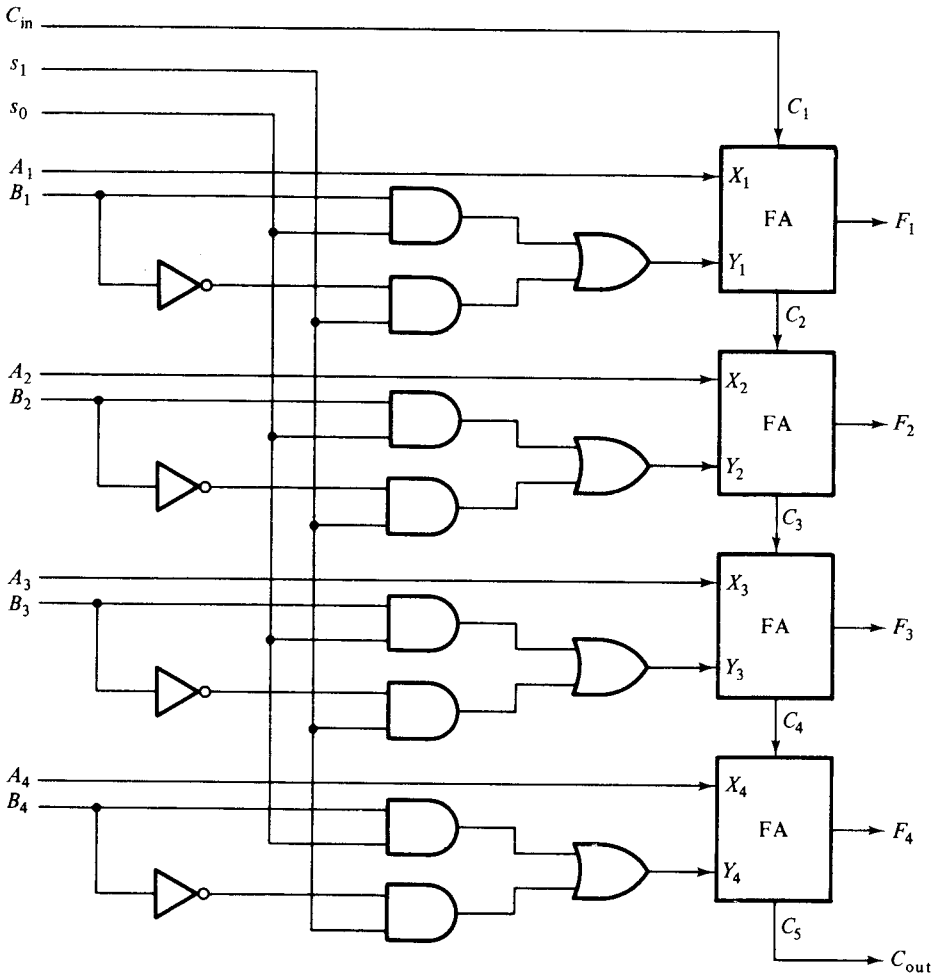


Figura 9-8 Diagrama lógico del circuito aritmético

Este ejemplo demuestra la factibilidad de construir un circuito aritmético por medio del sumador en paralelo. El circuito combinacional, que debe ser adicionado en cada etapa entre las entradas externas A_i y B_i y las entradas del sumador en paralelo X_i y Y_i, es una función de las operaciones aritméticas que van a ser configuradas. El circuito aritmético de la Figura 9-8 necesita un circuito combinacional para cada etapa especificada por las funciones de Boole:

$$X_i = A_i$$

$$Y_i = B_i s_0 + B_i' s_1 \qquad i = 1, 2, \ldots, n$$

donde n es el número de bits del circuito aritmético. En cada etapa i, se usan las mismas variables de selección común s_1 y s_0. El circuito combinacional será diferente si el circuito genera diferentes operaciones aritméticas.

Efecto del arrastre de salida

El arrastre de salida de un circuito aritmético o ALU tiene un significado especial, principalmente después de una operación de sustracción. Para investigar el efecto de un arrastre de salida, se expande el circuito aritmético de la Figura 9-8 a n bits de manera que $C_{out} = 1$, cuando la salida del circuito es igual o mayor que 2^n. La Tabla 9-2 lista las condiciones para tener un arrastre de salida en el circuito. La función $F = A$ tendrá siempre el arrastre de salida igual a 0. Lo mismo se aplica a la operación de incremento $F = A + 1$ excepto cuando pasa de una condición de sólo 1, a una condición de sólo 0, en cuyo tiempo se produce un arrastre de salida de 1. Un arrastre de salida de 1 después de una operación de adición denota una condición de sobrecapacidad. Este indica que la suma es mayor que o igual a 2^n y que la suma consiste de $n + 1$ bits.

La operación $F = A + \overline{B}$ agrega el complemento de 1 de $\overline{B}$ a A. Recuérdese de la Sección 1-5 que el complemento de B puede expresarse aritméticamente como $2^n - 1 - B$. El resultado aritmético de la salida será:

$$F = A + 2^n - 1 - B = 2^n + A - B - 1$$

Si $A > B$, entonces $(A - B) > 0$ y $F > (2^n - 1)$, de manera que $C = 1$. Quitando el arrastre de salida 2^n de este resultado dará:

$$F = A - B - 1$$

lo cual es una sustracción con bit prestado. Nótese que si $A \leqq B$, entonces $(A - B) \leqq 0$ y $F \leqq (2^n - 1)$ y así $C_{out} = 0$. Para esta condición es más conveniente expresar el resultado aritmético como:

$$F = (2^n - 1) - (B - A)$$

el cual es el complemento de 1 de $B - A$.

Tabla 9-2 Efecto del arrastre de salida en el circuito aritmético de la Figura 9-8

Selector de función			Función aritmética	$C_{out} = 1$ si	Comentarios
s_1	s_0	C_{in}			
0	0	0	$F = A$		C_{out} es siempre 0
0	0	1	$F = A + 1$	$A = 2^n - 1$	$C_{out} = 1$ y $F = 0$ si $A = 2^n - 1$
0	1	0	$F = A + B$	$(A + B) \geqslant 2^n$	Ocurre sobrecapacidad si $C_{out} = 1$
0	1	1	$F = A + B + 1$	$(A + B) \geqslant (2^n - 1)$	Ocurre sobrecapacidad si $C_{out} = 1$
1	0	0	$F = A - B - 1$	$A > B$	Si $C_{out} = 0$, entonces $A \leq B$ y F = complemento de 1 de $(B - A)$
1	0	1	$F = A - B$	$A \geqslant B$	Si $C_{out} = 0$, entonces $A < B$ y F = complemento de 2 de $(B - A)$
1	1	0	$F = A - 1$	$A \neq 0$	$C_{out} = 1$, excepto cuando $A = 0$
1	1	1	$F = A$		C_{out} es siempre 1

La condición para el arrastre de salida cuando $F = A + \bar{B} + 1$ puede deducirse de manera similar. $\bar{B} + 1$ es el símbolo para el complemento de 2 de B. Aritméticamente, ésta es una operación que produce un número igual a $2^n - B$. El resultado de la operación puede expresarse como:

$$F = A + 2^n - B = 2^n + A - B$$

Si $A \geqq B$, entonces $(A - B) \geqq 0$ y $F \geqq 2^n$, de manera que $C_{out} = 1$. Removiendo el arrastre de salida 2^n se obtiene:

$$F = A - B$$

la cual es una operación de sustracción. Si a pesar de que $A < B$ entonces $(A - B) < 0$ y $F < 2^n$ para que $C_{out} = 0$. El resultado aritmético para esta condición puede ser expresado como:

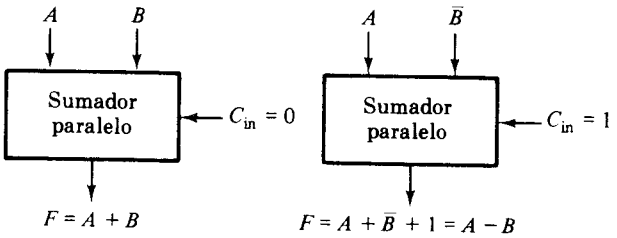
$$F = 2^n - (B - A)$$

lo cual es el complemento de 2 de $B - A$. Así, la salida de la sustracción aritmética es correcta siempre y cuando $A \geqq B$. La salida $B - A$ si $B > A$, pero el circuito genera el complemento de 2 de este número.

La operación de decremento se obtiene de $F = A + (2^n - 1) = 2^n + A - 1$. El arrastre de salida es siempre 1 excepto cuando $A = 0$. Sustrayendo 1 de 0 da -1 y -1 en complemento de 2 es $2^n - 1$ el cual es un número con sólo unos. La última entrada en la Tabla 9-2 genera $F = (2^n - 1) + A + 1 = 2^n + A$. Esta operación trasfiere A a F y da un arrastre de salida de 1.

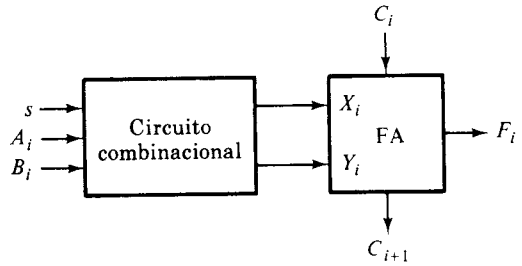
Diseño de otros circuitos aritméticos

El diseño de cualquier circuito aritmético que genera un conjunto de operaciones básicas puede llevarse a cabo siguiendo el procedimiento enunciado en el ejemplo previo. Asumiendo que todas las operaciones del grupo pueden ser generadas por medio del sumador en paralelo, se comienza obteniendo un diagrama de función como en la Figura 9-6. Del diagrama de función se obtiene una tabla de función que relaciona las entradas del circuito sumador completo a las entradas externas. A partir de la tabla de función se obtienen las compuertas combinacionales que deben ser agregadas a cada etapa del sumador completo. Este procedimiento se demuestra con el siguiente ejemplo.



(a) Especificación de la función

s	X_i	Y_i	C_{in}
0	A_i	B_i	0
1	A_i	B_i'	1



(b) Especificación del circuito combinacional

s	A_i	B_i	X_i	Y_i
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

$X_i = A_i$

$Y_i = B_i \oplus s$

$C_{in} = s$

(c) Tabla de verdad y ecuaciones simplificadas

Figura 9-9 Deducción de un circuito sumador/sustractor

EJEMPLO 9-1: Diseñar un circuito sumador/sustractor con una variable de selección s y dos entradas A y B. Cuando $s = 0$ el circuito realiza $A + B$. Cuando $s = 1$ el circuito ejecuta $A - B$ tomando el complemento de 2 de B.

La deducción del circuito aritmético se ilustra en la Figura 9-9. El diagrama de función se muestra en la Figura 9-9(a). Para la parte de suma, se necesita $C_{in} = 0$. Para la parte de sustracción se necesita el complemento de B y $C_{in} = 1$. La tabla de función se lista en la Figura 9-9(b). Cuando $s = 0$, X_i y Y_i de cada sumador completo deben ser iguales a las entradas externas A_i y B_i respectivamente. Cuando $s = 1$, se debe tener $X_i = A_i$ y $Y_i = B_i'$. El arrastre de salida debe ser igual al valor de s. El diagrama en (b) muestra la posición del circuito combinacional en una etapa típica del circuito aritmético. La tabla de verdad en (c) se obtiene listando los ocho valores de las variables de entrada binarias. La salida X_i se hace igual a la entrada A_i en ocho entradas. La salida Y_i es igual a B_i para las cuatro entradas cuando $s = 0$. Esta es igual al complemento de B_i para las últimas cuatro entradas cuando $s = 1$. Las funciones de salida simplificadas para los circuitos combinacionales son:

$$X_i = A_i$$
$$Y_i = B_i \oplus s$$

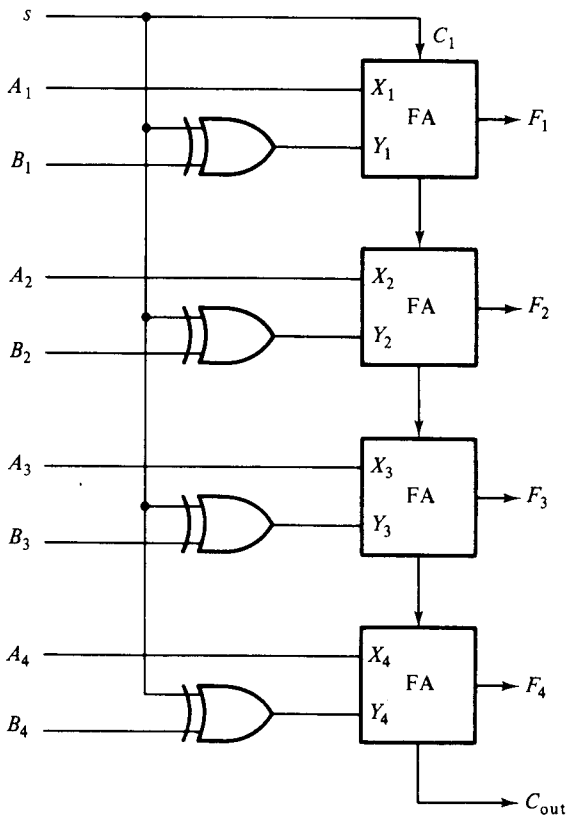
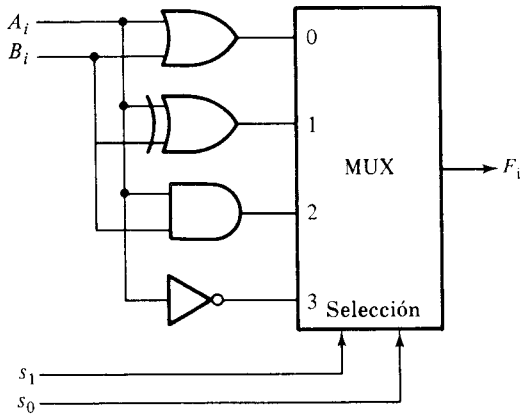


Figura 9-10 Circuito sumador/sustractor de 4 bits

El diagrama del circuito sumador sustractor de 4 bits se muestra en la Figura 9-10. Cada entrada B_i requiere una compuerta OR-exclusiva. La variable de selección s va a una entrada de cada compuerta y también al arrastre de entrada del sumador en paralelo. El sumador/sustractor de 4 bits puede ser construido con dos CI. Un CI es el sumador en paralelo de 4 bits y el otro es un CI de compuertas OR-exclusivas cuádruples.

9-5 DISEÑO DEL CIRCUITO LOGICO

Las microoperaciones lógicas manipulan los bits de los operandos separadamente y tratan cada bit como una variable binaria. La Tabla 2-6 lista 16 operaciones lógicas que pueden ser realizadas con dos variables binarias. Las 16 operaciones lógicas pueden ser generadas en un circuito y seleccionadas por medio de cuatro líneas de selección. Como todas las operaciones lógicas pueden obtenerse por medio de operaciones AND, OR y NOT (complemento), podría ser más conveniente emplear un circuito lógico justamente con esas operaciones. Para tres operaciones se necesitan dos variables de selección. Pero dos líneas de selección pueden seleccionar entre cuatro operaciones lógicas, de manera que se escoge también la función OR-ex-



(a) Diagrama lógico

s_1	s_0	Salida	Operacion
0	0	$F_i = A_i + B_i$	OR
0	1	$F_i = A_i \oplus B_i$	XOR
1	0	$F_i = A_i B_i$	AND
1	1	$F_i = A_i'$	NOT

(b) Tabla de función

Figura 9-11 Una etapa de un circuito lógico

clusiva (XOR) para el circuito lógico que va a diseñarse en esta y en la siguiente sección.

El método más simple y directo de diseñar un circuito lógico se muestra en la Figura 9-11. El diagrama muestra una etapa típica designada por el suscrito i. El circuito debe repetirse n veces para un circuito lógico de n bits. Las cuatro compuertas generan las cuatro operaciones lógicas OR, OR-exclusiva, AND y NOT. Las dos variables de selección en el multiplexor seleccionan una de las compuertas de la salida. La tabla de función lista la lógica de salida generada como una función de dos variables de selección.

El circuito lógico puede ser combinado en el circuito aritmético para producir una unidad lógica aritmética. Las variables de selección s_1 y s_0 pueden hacerse comunes a ambas secciones siempre y cuando se use una tercera variable de selección s_2 para diferenciar entre los dos. Esta configuración se ilustra en la Figura 9-12. Las salidas de los circuitos lógicos y aritméticos de cada estado pasan por un multiplexor con la variable de selección s_2. Cuando $s_2 = 0$ se selecciona la salida aritmética, pero cuando $s_2 = 1$ se selecciona la salida lógica. Aunque los dos circuitos pueden combinarse de esta manera, ésta no es la mejor forma de diseñar un ALU.

Un ALU más eficiente puede obtenerse si se investiga la posibilidad de generar operaciones lógicas de un circuito aritmético ya disponible. Esto puede hacerse inhibiendo todos los arrastres de entrada de los circuitos del sumador completo del sumador en paralelo. Considérese la función de Boole que genera la suma de salida de un circuito sumador completo:

$$F_i = X_i \oplus Y_i \oplus C_i$$

El arrastre de entrada C_i en cada etapa puede hacerse igual a 0 cuando la variable de selección s_2 sea igual a 1. El resultado será:

$$F_i = X_i \oplus Y_i$$

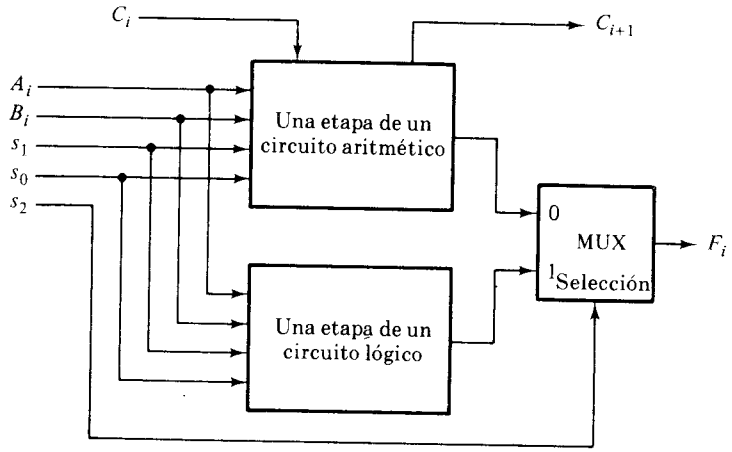


Figura 9-12 Combinando circuitos lógicos y aritméticos

Tabla 9-3 Operaciones lógicas en una etapa de un circuito aritmético

s_2	s_1	s_0	X_i	Y_i	C_i	$F_i = X_i \oplus Y_i$	Operación	Operación requerida
1	0	0	A_i	0	0	$F_i = A_i$	Trasferir A	OR
1	0	1	A_i	B_i	0	$F_i = A_i \oplus B_i$	XOR	XOR
1	1	0	A_i	B_i'	0	$F_i = A_i \odot B_i$	Equivalencia	AND
1	1	1	A_i	1	0	$F_i = A_i'$	NOT	NOT

Esta expresión es válida debido a la propiedad de la operación OR-exclusiva $x \oplus 0 = x$. Así, con el arrastre de salida de *cada* etapa igual a 0, los circuitos del sumador completo generan la operación OR-exclusiva.

Considérese el circuito aritmético de la Figura 9-8. El valor de Y_i puede seleccionarse por medio de dos variables de selección que sean iguales a 0, B_i, B_i' o 1. El valor de X_i es siempre igual a la entrada A_i. La Tabla 9-3 muestra las cuatro operaciones lógicas obtenidas cuando la tercera variable de selección $s_2 = 1$. Esta variable de selección obliga a que C_i sea igual a 0 mientras que s_1 y s_0 escogen un valor particular de Y_i. Las cuatro operaciones lógicas obtenidas por esta configuración son la trasferencia, la OR-exclusiva, la equivalencia y el complemento. La tercera entrada es la operación de equivalencia porque:

$$A_i \oplus B_i' = A_iB_i + A_i'B_i' = A_i \odot B_i$$

La última entrada en la tabla es el NOT u operación de complemento ya que:

$$A_i \oplus 1 = A_i'$$

La tabla tiene una columna más la cual contiene la lista de las cuatro operaciones lógicas que se van a incluir en el ALU. Dos de estas operaciones, la OR-exclusiva y el NOT están disponibles. La pregunta que debe ser contestada es de si es posible modificar el circuito aritmético de manera que genere las funciones lógicas OR y AND en vez de las funciones de trasferencia y equivalencia. Este problema se investiga en la siguiente sección.

9-6 DISEÑO DE UNA UNIDAD LOGICA ARITMETICA

En esta sección se diseña un ALU con ocho operaciones aritméticas y cuatro operaciones lógicas. Las tres variables de selección $s_2 s_1$ y s_0 seleccionan ocho operaciones diferentes y el arrastre de entrada C_{in} se usa para seleccionar cuatro operaciones aritméticas adicionales. Con $s_2 = 0$, las variables s_1 y s_0 conjuntamente con C_{in}, seleccionan las ochos operaciones aritméticas listadas en la Tabla 9-1. Con $s_2 = 1$, las variables s_1 y s_0 seleccionan las cuatro operaciones lógicas OR, OR-exclusiva, AND y NOT.

El diseño de un ALU es un problema de lógica combinacional. Debido a que la unidad tiene un patrón regular, ésta puede fraccionarse en etapas idénticas conectadas en cascada por medio de los arrastres. Se puede diseñar una etapa del ALU y luego duplicarla para conseguir el número de etapas requeridas. Hay seis entradas a cada etapa: A_i, B_i, C_i, s_2, s_1 y s_0. Hay dos salidas de cada etapa: la salida F_i y el arrastre de salida C_{i+1}. Se puede formular una tabla de verdad con 64 entradas y simplificar las dos funciones de salida. Aquí se escoge el uso de un procedimiento alterno que usa la disponibilidad de un sumador paralelo.

Los pasos de que se compone el diseño de un ALU son los siguientes:

1. Diseñar la sección aritmética independientemente de la sección lógica.
2. Determinar las operaciones lógicas obtenidas del circuito aritmético en el paso 1, asumiendo que los arrastres de salida de todas las etapas son 0.
3. Modificar el circuito aritmético para obtener las operaciones lógicas requeridas.

El tercer paso en el diseño no es un procedimiento directo y requiere cierta cantidad de ingenuidad por parte del diseñador. No hay garantía de que se pueda encontrar una solución o que la solución use el mismo número de compuertas. El ejemplo presentado aquí demuestra el tipo de pensamiento lógico que se requiere algunas veces en el diseño de sistemas digitales.

Se debe tener en cuenta que se dispone de varios ALU en CI encapsulados. En un caso práctico, lo que se debe hacer es buscar un ALU adecuado o unidad procesadora entre los circuitos integrados que se obtienen comercialmente. Pero, la lógica interna del CI seleccionado debe haber sido diseñado por una persona familiarizada con las técnicas de diseño lógico.

La solución para el primer paso del diseño se muestra en la Figura 9-8. La solución al segundo paso de diseño es presentado en la Tabla 9-3. La solución para el tercer paso se deduce a continuación.

De la Tabla 9-3 se observa que si $s_2 = 1$, el arrastre de entrada C_i en cada etapa debe ser 0. Con $s_1 s_0 = 00$ cada etapa así genera la función $F_i = A_i$. Para cambiar la salida a una operación OR, se debe cambiar la entrada a cada circuito sumador completo de A_i a $A_i + B_i$. Esto puede lograrse aplicando la función OR a B_i y A_i cuando $s_2 s_1 s_0 = 100$.

Las otras variables de selección que dan una salida indeseable ocurren cuando $s_2 s_1 s_0 = 110$. La unidad de esta manera genera una salida $F_i = A_i \odot B_i$ pero se requiere generar la operación AND $F_i = A_i B_i$. Se puede investigar la posibilidad de aplicar la función OR a cada entrada A_i con alguna función de Boole K_i. La función que se obtiene se usa para X_i cuando $s_2 s_1 s_0 = 110$:

$$F_i = X_i \oplus Y_i = (A_i + K_i) \oplus B_i' = A_i B_i + K_i B_i + A_i' K_i' B_i'$$

Una cuidadosa inspección del resultado revela que si la variable $K_i = B_i'$ se obtiene una salida:

$$F_i = A_i B_i + B_i' B_i + A_i B_i B_i' = A_i B_i$$

Dos términos son iguales a 0 porque $B_i B_i' = 0$. El resultado obtenido es la operación AND que se requiere. La conclusión es que, si A_i se aplica con B_i' a una función OR cuando $s_2 s_1 s_0 = 110$, la salida genera la operación AND.

El ALU final se muestra en la Figura 9-13. Solamente las dos etapas se dibujan, pero el diagrama puede extenderse fácilmente a más etapas. Las entradas a cada circuito sumador completo se especifican por medio de las funciones de Boole:

$$X_i = A_i + s_2 s_1' s_0' B_i + s_2 s_1 s_0' B_i'$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = s_2' C_i$$

Cuando $s_2 = 0$, las tres funciones se reducen a:

$$X_i = A_i$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$Z_i = C_i$$

las cuales son las funciones para el circuito aritmético de la Figura 9-8. Las operaciones lógicas se generan cuando $s_2 = 1$. Para $s_2 s_1 s_0 = 101$ ó 111, las funciones se reducen a:

$$X_i = A_i$$
$$Y_i = s_0 B_i + s_1 B_i'$$
$$C_i = 0$$

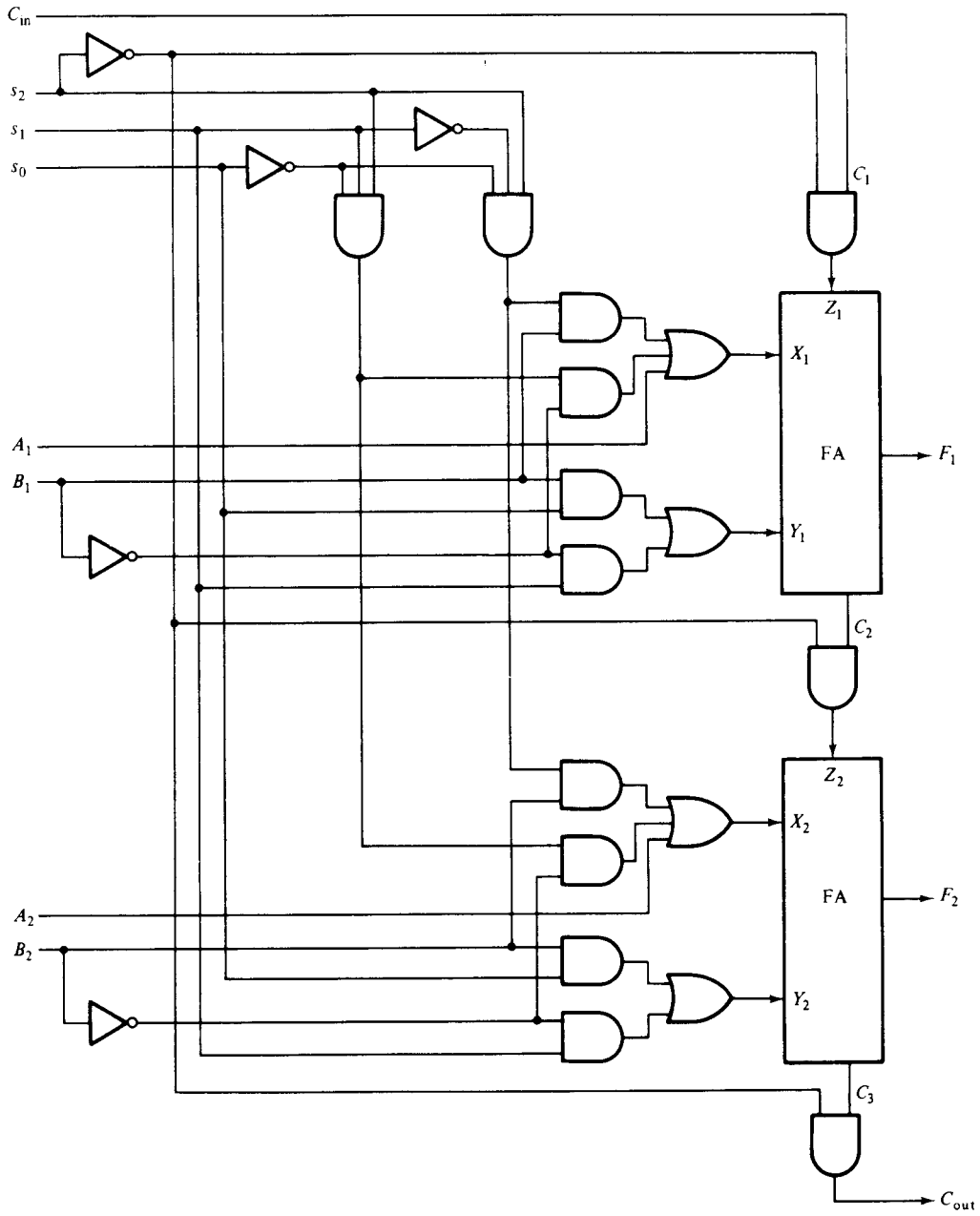


Figura 9-13 Diagrama lógico de una unidad lógica aritmética (ALU)

La salida F_i es igual a $X_i \oplus Y_i$ y produce las operaciones OR-exclusiva y de complemento como se especifica en la Tabla 9-3. Cada A_i con B_i se aplican a una función OR cuando $s_2 s_1 s_0 = 110$, para producir la operación OR como se ha discutido antes. Cada A_i con B_i' se aplican a una función OR cuando $s_2 s_1 s_0 = 110$ para producir una operación AND como se ha explicado previamente.

Las 12 operaciones generadas en el ALU se sumarizan en la Tabla 9-4. La función particular se selecciona por medio de s_2, s_1, s_0 y C_{in}. Las operaciones aritméticas son idénticas a aquellas listadas para el circuito aritmético. El valor de C_{in} para las cuatro funciones lógicas no tienen efecto en la operación de la unidad y aquellas entradas se marcan con X de no importa.

Tabla 9-4 Tabla de función para el ALU de la Figura 9-13

Selección					
s_2	s_1	s_0	C_{in}	Salida	Función
0	0	0	0	$F = A$	Trasferir A
0	0	0	1	$F = A + 1$	Incrementar A
0	0	1	0	$F = A + B$	Suma
0	0	1	1	$F = A + B + 1$	Suma con arrastre
0	1	0	0	$F = A - B - 1$	Resta con préstamo
0	1	0	1	$F = A - B$	Sustracción
0	1	1	0	$F = A - 1$	Decrementar A
0	1	1	1	$F = A$	Trasferir A
1	0	0	X	$F = A \vee B$	OR
1	0	1	X	$F = A \oplus B$	OR-exclusiva
1	1	0	X	$F = A \wedge B$	AND
1	1	1	X	$F = \bar{A}$	Complementar A

9-7 REGISTRO DE CONDICION

Las magnitudes relativas de dos números pueden ser determinadas restando un número de otro y luego combinando ciertas condiciones de los bits en la diferencia resultante. Si los dos números están sin signo las condiciones de los bits de algún interés, son el arrastre de salida y un resultado posible de cero. Si los dos números incluyen un bit de signo en la posición de mayor orden, las condiciones principales de los bits, son el signo del resultado, una indicación de cero y una condición de sobrecapacidad. Es conveniente algunas veces suplementar el ALU con un registro de condición donde se almacenan aquellas condiciones de los bits para análisis posterior. El estado de los bits de condición se llama algunas veces *código de condición* de los bits o bits *indicadores*.

La Figura 9-14 muestra un diagrama de bloque de un ALU de 8 bits con un registro de condición de 4 bits. Los cuatro bits de condición se simbolizan por medio de C, S, Z y V. Los bits se ponen a uno ó cero como resultado de una operación realizada en el ALU.

1. El bit C se pone a uno si el arrastre de salida del ALU es 1 y se pone a cero (borrado) si el arrastre de salida es 0.
2. El bit S se pone a uno si el bit de mayor orden del resultado en la salida del ALU (bit del signo) es 1 y se pone a cero (borrado) si el bit de mayor orden es 0.
3. El bit Z se pone a uno si la salida del ALU contiene sólo ceros y se pone a cero (borrado) de otra manera. $Z = 1$ si el resultado es cero y $Z = 0$ si el resultado es diferente de cero.
4. El bit V se pone a uno si la OR-exclusiva de los arrastres C_8 y C_9 es 1 y de otra manera se pone a cero (borrado). Esta es la condición de sobrecapacidad cuando los números están en la representación de signo-complemento de 2 (ver Sección 8-5). Para el ALU de 8 bits, V se pone a uno si el resultado es mayor que 127 o menor que -128.

Los bits de condición pueden comprobarse después de una condición de operación para determinar ciertas relaciones que existen entre los valores de A y B. Si el bit V se pone a uno después de la adición de los dos números con signo, éste indicará una condición de sobrecapacidad. Si Z se pone a uno después de una operación de OR-exclusiva indica que $A = B$. Esto es así porque $x \oplus x = 0$ y la OR-exclusiva de dos operandos iguales da un resultado de sólo ceros, los cuales ponen a uno el bit Z. Un solo bit de A puede comprobarse para determinar si es 0 ó 1, al enmascarar todos los bits excepto el bit en prueba, para luego comprobar el bit de condición Z.

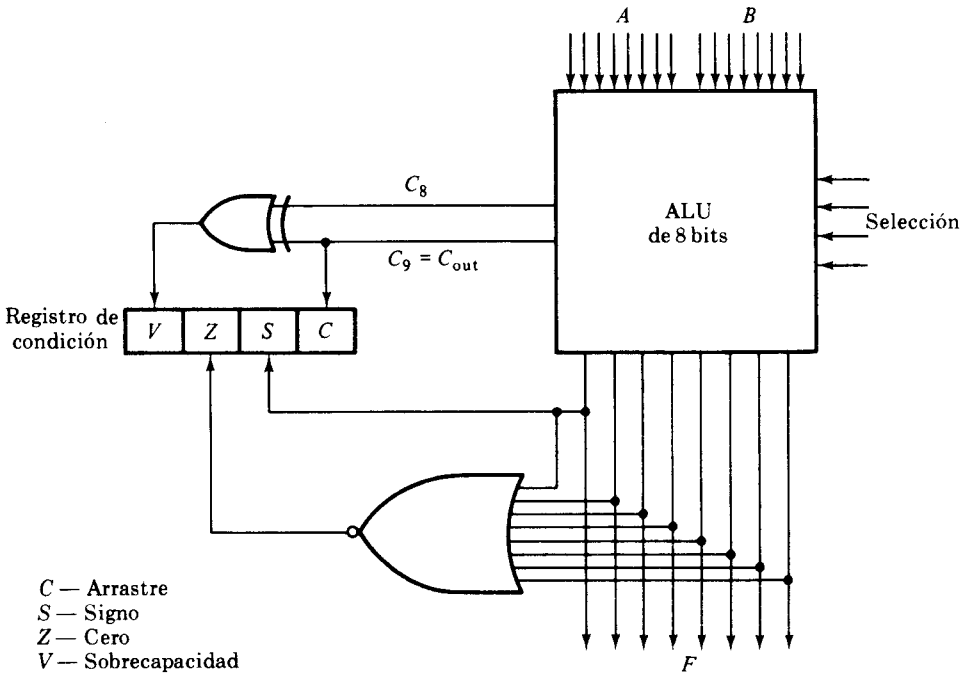


Figura 9-14 Activación de los bits en un registro de estado

Por ejemplo, sea $A = 101x1100$ donde x es el bit que se va a comprobar. La operación AND de A con $B = 00010000$ producirá un resultado $000x0000$. Si $x = 0$ el bit de condición se pone a uno pero si $x = 1$ el bit Z se borra ya que el resultado es cero.

La operación de *comparación* es una sustracción de A menos B, excepto que el resultado de la operación no se trasfiere al registro de destino, pero los bits de condición se afectan. El registro de condición suministra entonces la información acerca de las magnitudes relativas de A y B. Los bits de condición que se van a considerar dependen de si se toman los dos números con signo o sin él en la representación de complemento de 2.

Considérese la operación $A - B$ hecha con dos números binarios *sin signo*. Las magnitudes relativas de A y B pueden determinarse de los valores trasferidos a los bits de condición C y Z. Si $Z = 1$ se sabe que $A = B$ ya que $A - B = 0$. Si $Z = 0$ se sabe que $A \neq B$. De la Tabla 9-2 se tiene que $C = 1$ se tiene que $C = 1$ si $A \geq B$ y $C = 0$ si $A < B$. Estas condiciones están listadas en la Tabla 9-5. La tabla da en su lista otras dos condiciones. Para que A sea mayor pero no igual a B $(A > B)$ se debe tener $C = 1$ y $Z = 0$. Como C se pone a uno cuando el resultado es 0, se debe comprobar Z para asegurarse que el resultado no es 0. Para que A sea menor que o igual a $B (A \leq B)$, el bit C debe ser 0 (para $A < B$) o el bit Z debe ser 1 (para $A = B$). La Tabla 9-5 lista también las funciones de Boole que deben satisfacerse para cada una de las seis relaciones.

Algunos computadores consideran el bit C como el bit de préstamo después de una operación de sustracción de $A - B$. Un bit de préstamo no ocurre si $A \geq B$ pero un bit extra debe ser prestado cuando $A < B$. La condición para el bit de préstamo es el complemento del arrastre de salida obtenido cuando se hace la sustracción, tomando el complemento de 2 de B. Por esta razón un procesador que considera el bit C como el bit de préstamo después de una sustracción, complementará el bit C después de la sustracción u operación de comparación y denotará este bit como préstamo.

Considérese ahora la operación $A - B$ hecha con dos números binarios *con signo* cuando los números negativos están en la forma de complemento de 2. Las magnitudes relativas de A y B pueden ser determinadas de dos valores trasferidos a los bits de condición Z, S y V. Si $Z = 1$ se conoce que $A = B$, cuando $Z = 0$ se tiene que $A \neq B$. Si $S = 0$, el signo del resultado es

Tabla 9-5 Bits de condición después de la sustracción de los números sin signo $(A - B)$

Relación	Estado del bit de condición	Función de Boole
$A > B$	$C = 1$ y $Z = 0$	CZ'
$A \geq B$	$C = 1$	C
$A < B$	$C = 0$	C'
$A \leq B$	$C = 0$ o $Z = 1$	$C' + Z$
$A = B$	$Z = 1$	Z
$A \neq B$	$Z = 0$	Z'

positivo de manera que A debe ser mayor que B. Esto es verdad si no hay sobrecapacidad y $V = 0$. Si el resultado se desborda por sobrecapacidad se obtendrá un resultado erróneo. Fue mostrado en la Sección 8-5 que una condición de sobrecapacidad cambia el signo del resultado. Por tanto, si $S = 1$ y $V = 1$ esto indica que el resultado debería haber sido positivo y por tanto A debe ser mayor que B.

La Tabla 9-6 lista las seis relaciones posibles que pueden existir entre A y B y los valores correspondientes de Z, S y V en cada caso. Para que $A - B$ sea mayor que pero no igual a cero ($A > B$), el resultado debe ser positivo o diferente de cero. Como un resultado de cero dará un signo positivo, se debe asegurar que el bit Z es 0 para excluir la posibilidad de $A = B$. Para $A \geq B$ es suficiente comprobar un signo positivo cuando no ocurre sobrecapacidad o un signo negativo cuando ocurre una sobrecapacidad. Para $A < B$, el resultado debe ser negativo. Si el resultado es negativo o cero, se tiene que $A \leq B$. Las funciones de Boole listadas en la tabla expresan las condiciones del bit de condición en forma algebraica.

Tabla 9-6 Bits de condición después de la sustracción de números $(A - B)$ en signo-complemento de 2

Relación	Estado de los bits de condición	Función de Boole
$A > B$	$Z = 0$ y $(S = 0, V = 0$ ó $S = 1, V = 1)$	$Z'(S \odot V)$
$A \geq B$	$S = 0, V = 0$ ó $S = 1, V = 1$	$S \odot V$
$A < B$	$S = 1, V = 0$ ó $S = 0, V = 1$	$S \oplus V$
$A \leq B$	$S = 1, V = 0$ ó $S = 0, V = 1$ ó $Z = 1$	$(S \oplus V) + Z$
$A = B$	$Z = 1$	Z
$A \neq B$	$Z = 0$	Z'

9-8 DISEÑO DE UN REGISTRO DE DESPLAZAMIENTO

La unidad de desplazamiento adjunta a un procesador trasfiere la salida del ALU al bus de salida. La unidad de desplazamiento puede trasferir la información directamente sin un desplazamiento o puede desplazar la información a la derecha o a la izquierda. Se debe tener alguna precaución para que algunas veces no haya trasferencia del ALU al bus de salida. El registro de desplazamiento produce la microoperación de desplazamiento comúnmente no disponible en un ALU.

Un circuito obvio para un registro de desplazamiento es un registro de desplazamiento bidireccional con carga en paralelo. La información del ALU puede ser trasferida al registro en paralelo para luego desplazarla a la derecha o a la izquierda. En esta configuración se necesita un pulso de reloj para la trasferencia al registro de desplazamiento y se necesita otro pulso para el desplazamiento. Estos dos pulsos son agregados al pulso necesario para trasferir la información del registro de desplazamiento al registro de destino.

La trasferencia de un registro fuente a un registro de destino puede hacerse con un pulso de reloj si se configura el registro de desplazamiento con un circuito combinacional. En un registro de desplazamiento de lógica combinacional, las señales del ALU al bus de salida se propagarán por las compuertas sin la necesidad de un pulso de reloj. Por tanto el único pulso de reloj necesario en el sistema del procesador es para cargar los datos del bus de salida al registro de destino.

Un registro de desplazamiento de lógica combinacional puede construirse con multiplexores como se muestra en la Figura 9-15. Las dos variables de selección H_1 y H_0 aplicadas a los cuatro multiplexores seleccionan el tipo de operación en el registro de desplazamiento. Con $H_1H_0 = 00$ no se ejecutan desplazamientos y las señales de F van directamente a las líneas de S. Las dos siguientes variables de selección causan una operación de desplazamiento a la derecha o la izquierda. Cuando $H_1H_0 = 11$, los multiplexores seleccionan las entradas conectadas a 0 y como una secuencia las salidas de S son también iguales a 0, bloqueando la trasferencia de información del ALU al bus de salida. La Tabla 9-7 sumariza la operación del registro de desplazamiento.

El diagrama de la Figura 9-15 muestra solamente cuatro estados del registro de desplazamiento. Este último por supuesto debe consistir de n estados en un sistema con n líneas en paralelo. Las entradas I_R e I_L sirven como entradas de serie para la primera y última etapas durante un desplazamiento a la derecha o a la izquierda respectivamente. Otra variable de selección puede ser empleada para especificar que irá a I_R e I_L durante el desplazamiento. Por ejemplo, una tercera variable de selección H_2, cuando está en un estado puede seleccionar un 0 para la entrada en serie

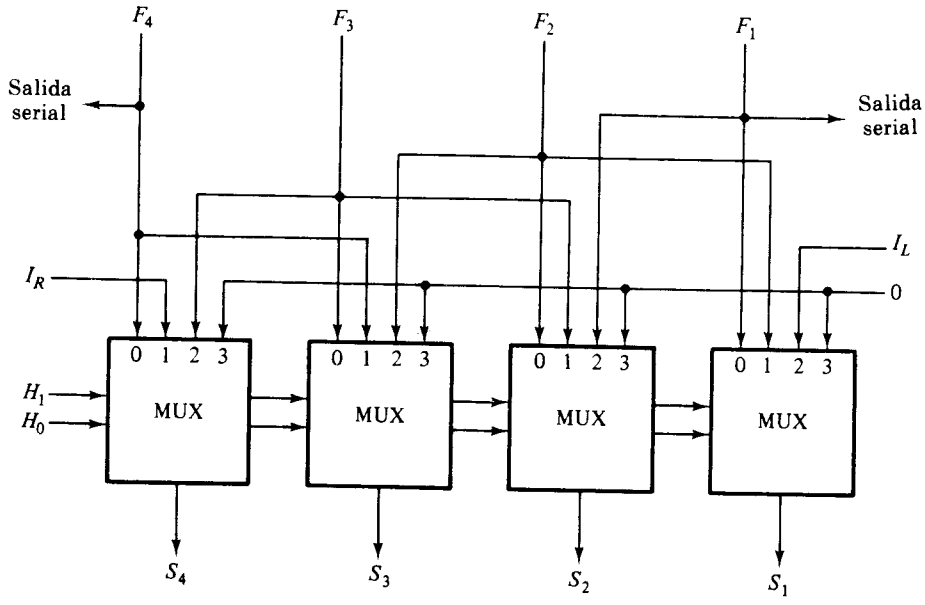


Figura 9-15 Registro de desplazamiento de 4 bits a base de lógica combinacional

Tabla 9-7 Tabla de función para el registro de desplazamiento

H_1	H_0	Operación	Función
0	0	$S \leftarrow F$	Trasferir F a S (ningún desplazamiento)
0	1	$S \leftarrow \text{shr } F$	Desplazar F a la derecha hacia S
1	0	$S \leftarrow \text{shl } F$	Desplazar F a la izquierda hacia F
1	1	$S \leftarrow 0$	Trasferir 0 a S

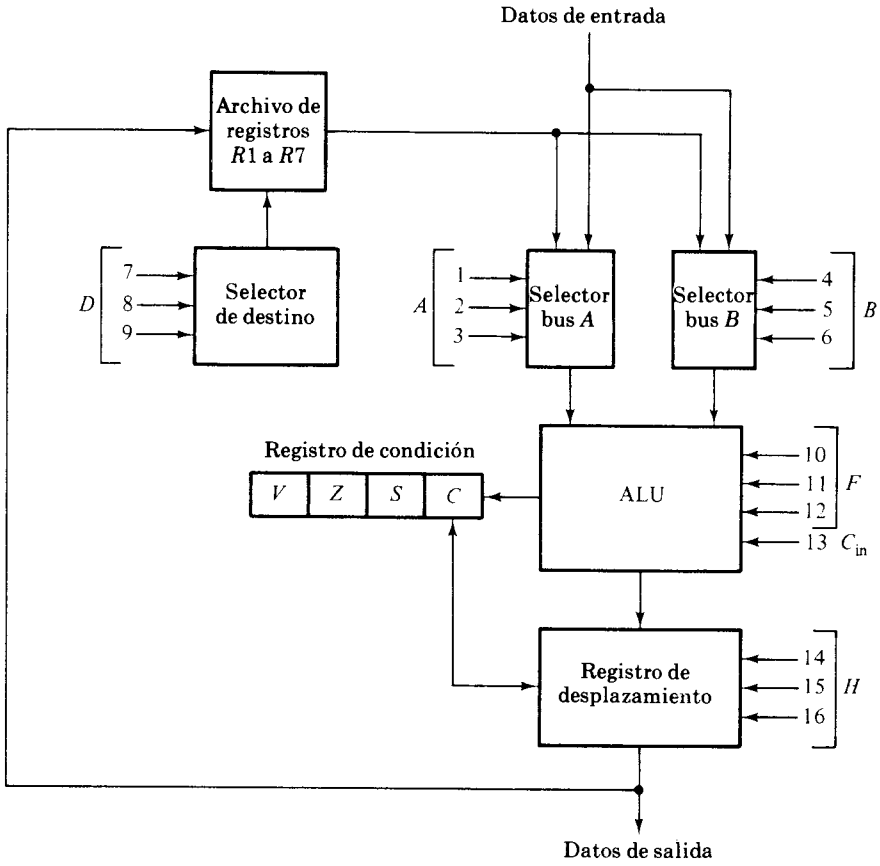
durante el desplazamiento. Cuando H_2 está en el otro estado la información puede circularse conjuntamente con el valor del bit de condición. De esta manera un arrastre producido durante una operación de suma puede desplazarse a la derecha a la posición del bit más significativa de un registro.

9-9 UNIDAD PROCESADORA

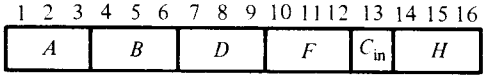
Las variables de selección en la unidad procesadora controla las microoperaciones ejecutadas dentro del procesador durante cualquier pulso de reloj dado. Las variables de selección controlan los buses, el ALU, el registro de desplazamiento y el registro de destino. Se demostrará ahora por medio de un ejemplo cómo las variables de control seleccionan las microoperaciones en una unidad procesadora. El ejemplo define una unidad procesadora conjuntamente con todas las variables de selección. Luego se discutirán las alternativas de las variables de control para algunas microoperaciones típicas.

Un diagrama de bloque de una unidad procesadora se muestra en la Figura 9-16(a). Este consiste de siete registros $R1$ hasta $R7$ y el registro de condición. Las salidas de los siete registros van a través de dos multiplexores para seleccionar las entradas del ALU. La entrada de datos de una fuente externa se selecciona también con los mismos multiplexores. La salida del ALU pasa a través de un registro de desplazamiento y luego va a un grupo de terminales de salida externos. La salida del registro de desplazamiento puede trasferirse a cualquiera de los registros o a un destino externo.

Hay 16 variables de selección en la unidad y su función se especifica por una *palabra de control* en la Figura 9-16(b). La palabra de control de 16 bits cuando se aplica a las variables de selección en el procesador, especifica una microoperación dada. La palabra de control se divide en seis campos, con cada campo designado por una letra. Todos los campos, excepto C_{in} tienen un código de tres bits. Los tres bits de A seleccionan un registro fuente para la entrada en la parte izquierda del ALU. El campo B es el mismo, pero selecciona la fuente de información para la entrada derecha del ALU. El campo D selecciona un registro de destino. El campo F conjuntamente con el bit en C_{in} seleccionan una función en el ALU. El campo H selecciona el tipo de desplazamiento y el registro de desplazamiento.



(a) Diagrama de bloque



(b) Palabra de control

Figura 9-16 Unidad procesadora con variables de control

Las funciones de todas las variables de selección se especifican en la Tabla 9-8. El código binario de 3 bits listado en la tabla especifica el código para cada uno de los cinco campos *A*, *B*, *D*, *F* y *H*. El registro seleccionado por *A*, *B* y *D* es aquel cuyo número decimal es equivalente al número binario en el código. Cuando el campo *A* ó *B* es 000, el correspondiente multiplexor selecciona la entrada de datos. Cuando $D = 000$, no se selecciona registro de destino. Los tres bits en el campo *F* conjuntamente con el arrastre de entrada C_{in}, sumunistran las 12 operaciones del ALU de la manera espe-

cificada en la Tabla 9-4. Nótese que hay dos posibilidades para $F = A$. En un caso el bit de arrastre C se borra y en el otro caso se pone a 1 (ver Tabla 9-2).

Tabla 9-8 Funciones de las variables de control para el procesador de la Figura 9-16

Código binario	Función de las variables de selección					
	A	B	D	F con $C_{in} = 0$	F con $C_{in} = 1$	H
0 0 0	Datos entr.	Datos sal.	Ning.	$A, C \leftarrow 0$	$A + 1$	No desplazamiento
0 0 1	$R1$	$R1$	$R1$	$A + B$	$A + B + 1$	Despl. a la der., $I_R = 0$
0 1 0	$R2$	$R2$	$R2$	$A - B - 1$	$A - B$	Despl. a la izq., $I_L = 0$
0 1 1	$R3$	$R3$	$R3$	$A - 1$	$A, C \leftarrow 1$	0 al bus de salida
1 0 0	$R4$	$R4$	$R4$	$A \vee B$	—	—
1 0 1	$R5$	$R5$	$R5$	$A \oplus B$	—	Circular a la der. con C
1 1 0	$R6$	$R6$	$R6$	$A \wedge B$	—	Circular a la izq. con C
1 1 1	$R7$	$R7$	$R7$	$\overline{A}$	—	—

Las cuatro primeras entradas del código en el campo H especifican las operaciones de desplazamiento de la Tabla 9-7. Una tercera variable de selección se usa para especificar un 0 para las entradas de serie I_R e I_L ó un desplazamiento circular con el bit de arrastre C. Por conveniencia se designa un desplazamiento circular a la derecha con arrastre como *crc* y a la izquierda como *clc*. Entonces la declaración (*crc* = circular right-shift with carry; *clc* = circular left-shift with carry):

$$R \leftarrow \text{crc } R$$

es una abreviación de la proposición:

$$R \leftarrow \text{shr } R, \qquad R_n \leftarrow C, \qquad C \leftarrow R_1$$

R se desplaza a la derecha, su bit menos significativo R_1 va a C y el valor de C va a la posición del bit más significativo R_n.

Se necesita una palabra de control de 16 bits para especificar una microoperación para la unidad de proceso. La manera más eficiente de generar palabras de control con tantos bits es almacenarlas en una unidad de memoria que funciona como *memoria de control* donde están almacenadas todas las palabras de control. La secuencia de palabras de control se lee de la memoria de control palabra a palabra para iniciar la secuencia deseada de microoperaciones. Este tipo de organización de control se llama *microprogramación* y se discute en más detalle en el Capítulo 10.

La palabra de control para una microoperación dada puede ser derivada directamente de las variables de selección definidas en la Tabla 9-8. La microoperación de sustracción:

$$R1 \leftarrow R1 - R2$$

especifica $R1$ para la entrada izquierda del ALU, $R2$ para la entrada derecha del ALU. $A - B$ para la operación del ALU, ningún desplazamiento para el registro de desplazamiento y $R1$ para el registro de destino. De la Tabla 9-8 se deriva la palabra de control para que esta operación sea

0010100010101000:

A	B	D	F	C_{in}	H
001	010	001	010	1	000

Las palabras de control para esta microoperación y algunas otras se listan en la Tabla 9-9.

La operación de *comparar* es similar a la microoperación de sustracción, excepto que la diferencia no se trasfiere al registro de destino y solamente los bits de condición se afectan. El campo de destino D en este caso debe ser 000. La trasferencia de $R4$ a $R5$ requiere una operación ALU $F = A$. La fuente A es 100 y el destino D es 101. El código de selección B podría ser cualquier cosa porque el ALU no lo usa. Este campo se marca con 000 en la tabla por conveniencia pero cualquier otro código de 3 bits podría ser usado.

Para trasferir la entrada de datos a $R6$, se debe tener $A = 000$ para seleccionar la entrada externa y $D = 110$ para seleccionar el registro de destino. De nuevo el valor de B no importa y la función ALU es $F = A$. Para sacar datos de $R7$ se hace $A = 111$ y $D = 000$ (ó 111). La operación de ALU $F = A$ coloca la información de $R7$ al bus de salida.

Es necesario algunas veces borrar o poner a 1 el bit de arrastre antes de una operación circular de desplazamiento. Esto puede hacerse con un código de selección de ALU 0000 ó 0111. Con el primer código se tiene el bit C borrado y con el segundo el bit C en 1. La trasferencia $R1 \leftarrow R1$, $C \leftarrow 0$ no cambia el contenido del registro, pero borrará C y V. Los bits de selección A y S se afectan de igual manera. Si $R1 = 0$ entonces Z se pone a 1, de otra manera se borrará. El bit S se pone a uno con el valor del bit signo en $R1$.

El pulso de reloj que dispara el registro de destino trasfiere también los bits de condición del ALU al registro de condición. Los bits de condición

Tabla 9-9 Ejemplos de microoperaciones para un procesador

Microoperación	Palabra de control						Función
	A	B	D	F	C_{in}	H	
$R1 \leftarrow R1 - R2$	001	010	001	010	1	000	Sustraer $R2$ de $R1$
$R3 - R4$	011	100	000	010	1	000	Comparar $R3$ y $R4$
$R5 \leftarrow R4$	100	000	101	000	0	000	Trasferir $R4$ a $R5$
$R6 \leftarrow$ Entrada	000	000	110	000	0	000	Entrada de datos a $R6$
Salida $\leftarrow R7$	111	000	000	000	0	000	Salida de datos de $R7$
$R1 \leftarrow R1$, $C \leftarrow 0$	001	000	001	000	0	000	Borrar el bit de arrastre C
$R3 \leftarrow$ shl $R3$	011	011	011	100	0	010	Desplazar a la derecha $R3$ con $I_L = 0$
$R1 \leftarrow$ crc $R1$	001	001	001	100	0	101	Circular a la der. $R1$ con bit de arrastre
$R2 \leftarrow 0$	000	000	010	000	0	011	Borrar $R2$

son afectados después de las operaciones aritméticas. Los bits de condición C y V se dejan sin cambiar durante una operación lógica ya que esos bits no tienen significado para las operaciones lógicas. En algunos procesadores es costumbre no cambiar el valor del bit de arrastre C después de una operación de incremento o decremento.

Si se quiere colocar el contenido de un registro a un registro de desplazamiento sin cambiar el bit de arrastre, se puede usar la operación lógica OR con el mismo registro seleccionado para las entradas A y B del ALU. La operación:

$$R \leftarrow R \vee R$$

no cambia el valor del registro R, sin embargo, coloca el contenido de R a las entradas del registro de desplazamiento y *no cambia* los valores de los bits de condición C y V.

Los ejemplos en la Tabla 9-9 discutidos hasta ahora usan el código 000 de selección de desplazamiento para el campo H, para indicar una operación de no desplazamiento. Para desplazar los contenidos de un registro se debe colocar el valor del registro en el registro de desplazamiento sin hacerle ningún cambio al ALU. La proposición de la microoperación de desplazamiento a la izquierda:

$$R3 \leftarrow \text{shl } R3$$

especifica el código para la selección de desplazamiento pero no el código para el ALU. El contenido de $R3$ puede colocarse en el registro de desplazamiento para especificar una operación OR entre $R3$ y el registro mismo. La información desplazada regresa a $R3$ si $R3$ se especifica como el registro de destino. Esto requiere que los campos de selección A, B y D tengan el código 011 para $R3$, que el código de función del ALU sea 1000 para la operación OR y que el selector H de desplazamiento sea 010 para el desplazamiento a la izquierda.

El desplazamiento circular a la derecha con arrastre del registro $R1$ se simboliza por medio de la proposición:

$$R1 \leftarrow \text{crc } R1$$

Esta proposición especifica el código del registro de desplazamiento pero no el código para el ALU. Para colocar el contenido de $R3$ en los terminales de salida del ALU sin afectar el bit C se usa la operación OR como se hizo anteriormente. De esta manera no se afecta el bit C en la operación del ALU, pero puede cambiarse debido al desplazamiento circular.

El último ejemplo en la Tabla 9-9 muestra la palabra de control para borrar un registro a 0. Para borrar el registro $R2$, se hace que el bus de salida contenga sólo ceros con $H = 011$. El campo de destino D se hace igual al código del registro $R2$.

Es obvio a partir de estos ejemplos que muchas otras microoperaciones pueden generarse en la unidad procesadora. Una unidad procesadora con un conjunto completo de microoperaciones es un elemento de propósito ge-

neral que puede adaptarse a muchas aplicaciones. El método de trasferencia entre registros es una herramienta conveniente para especificar las operaciones en forma simbólica en un sistema digital que emplea una unidad de proceso para propósitos generales. El sistema se define primero con una secuencia de proposiciones de microoperación en el método de notación de trasferencia entre registros o en cualquier otra notación disponible de trasferencia. Una función de control se representa aquí no por una función de Boole sino por una cadena de variables binarias llamadas palabra de control. La palabra de control para cada microoperación se deriva de la tabla de función del procesador.

La secuencia de palabras de control para el sistema se almacena en la memoria de control. La salida de la memoria de control se aplica a las variables de selección del procesador. Leyendo consecutivamente las palabras de control de la memoria, es posible darle secuencia a las microoperaciones del procesador. Así, todo el diseño puede hacerse por medio del método de trasferencia entre registros, el cual en este caso particular se refiere al *método de la microprogramación*. Este método para controlar la unidad procesadora se demuestra en la Sección 10-5.

9-10 DISEÑO DEL ACUMULADOR

Algunas unidades procesadoras distinguen un registro de otros y lo llaman registro acumulador. La organización de una unidad procesadora con un registro acumulador se muestra en la Figura 9-4. El ALU asociado con el registro puede ser construido como un circuito combinacional del tipo discutido en la Sección 9-5. En esta configuración, el registro acumulador es esencialmente un registro de desplazamiento bidireccional con carga en paralelo el cual es conectado a un ALU. Debido a la conexión de retroalimentación de la salida del registro a una de las entradas en el ALU el registro acumulador y su lógica asociada, cuando se toman como una sola unidad, constituyen un circuito secuencial. Debido a la anterior propiedad un registro acumulador puede ser designado por técnicas de circuitos secuenciales en vez de usar un ALU de circuito combinacional.

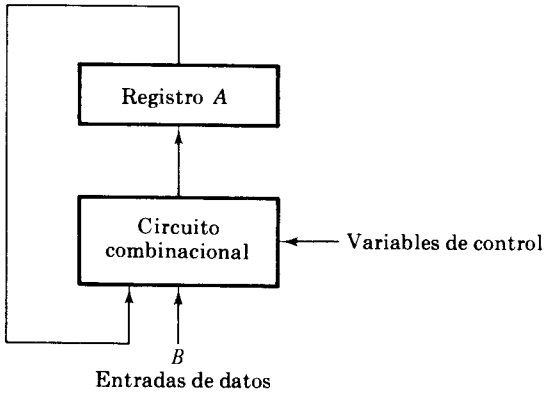


Figura 9-17 Diagrama de bloque del acumulador

El diagrama de bloque de un acumulador que forma un circuito secuencial se muestra en la Figura 9-17. El registro A y el circuito combinacional asociado constituyen un circuito secuencial. El circuito combinacional remplaza al ALU pero no puede separarse del registro ya que éste es solamente la parte del circuito combinacional del circuito secuencial. El registro A se trata algunas veces como el registro acumulador y se denota algunas veces por el símbolo AC. Aquí, el acumulador se refiere al registro A y a un circuito combinacional asociado. Las entradas externas al acumulador son las entradas de datos de B y las variables de control que determinan las microoperaciones del registro. El siguiente estado del registro A es una función de un estado presente y de las entradas externas.

En el Capítulo 7 se consideraron varios registros que realizan funciones específicas tales como la carga en paralelo, las operaciones de desplazamiento y el conteo. El acumulador es similar a estos registros pero es más general ya que éste puede realizar no solamente las funciones anteriores sino también otras operaciones de procesamiento de datos. Un acumulador es un registro de multifunción que por sí mismo puede realizar todas las microoperaciones en la unidad procesadora. Las microoperaciones incluidas en un acumulador dependen de las operaciones que deben incluirse en el procesador particular. Para demostrar el diseño lógico de un registro operacional de multipropósito tal como un acumulador se diseña el circuito con nueve microoperaciones. El procedimiento enunciado en esta sección puede ser usado para extender el registro a otras microoperaciones.

El conjunto de microoperaciones para el acumulador se da en la Tabla 9-10. Las variables de control p_1 hasta p_9 son generadas por los circuitos lógicos de control y deben ser consideradas como funciones de control que inician las operaciones correspondientes de trasferencia entre registros. El registro A es un registro fuente en todas las microoperaciones listadas. En esencia éste representa el estado presente del circuito secuencial. El registro B se usa como un segundo registro fuente para microoperaciones que necesitan dos operandos. El registro B se asume que está conectado al

Tabla 9-10 Lista de microoperaciones de un acumulador

Variable de control	Microoperación	Nombre
p_1	$A \leftarrow A + B$	Sumar
p_2	$A \leftarrow 0$	Borrar
p_3	$A \leftarrow \overline{A}$	Complementar
p_4	$A \leftarrow A \wedge B$	AND
p_5	$A \leftarrow A \vee B$	OR
p_6	$A \leftarrow A \oplus B$	OR-exclusiva
p_7	$A \leftarrow \text{shr } A$	Desplazar a la derecha
p_8	$A \leftarrow \text{shl } A$	Desplazar a la izquierda
p_9	$A \leftarrow A + 1$	Incremento
	Si $(A = 0)$ entonces $(Z = 1)$	Comprobar el cero

acumulador y suministra las entradas al circuito secuencial. El registro de destino para todas las microoperaciones es siempre el registro A. La nueva información trasferida a A constituye el siguiente estado del circuito secuencial. Las nueve variables de control se consideran también como entradas al circuito secuencial. Estas variables son excluyentes mutuamente y solamente una variable debe ser habilitada cuando ocurre un pulso de reloj. La última entrada en la Tabla 9-10 es una declaración de control condicional. Esta produce un 1 binario en una variable de salida Z, cuando el contenido del registro A es 0, es decir, cuando los flip-flops del registro están borrados.

Procedimiento de diseño

El acumulador consiste de n etapas y n flip-flops $A_1, A_2, \ldots, A_n$ numeradas consecutivamente comenzando desde la posición de la extrema izquierda. Es conveniente la partición del acumulador en n etapas similares compuesta cada una de un flip-flop denotado como A_i, una entrada de datos denotada por B_i y la lógica combinacional asociada con el flip-flop. En el procedimiento de diseño que sigue se considera solamente una etapa típica i teniendo en cuenta que un acumulador de n bits consiste de n etapas para $i = 1, 2, \ldots, n$. Cada etapa A_i se interconecta a su etapa A_{i-1} vecina a su derecha y a la A_{i+1} a su izquierda. La primera etapa A_1 y la última no tienen vecinas a un lado y debe prestárseles atención especial. El registro se diseñará usando flip-flops del tipo JK.

Cada variable de control p_j, $j = 1, 2, \ldots, 9$ inicia una microoperación particular. Para que la operación tenga significado se debe asegurar que solamente una variable de control se habilita en un tiempo dado. Como las variables de control son excluyentes mutuamente, es posible separar el circuito combinacional de una etapa en circuitos menores, uno para cada microoperación. Así, el acumulador se debe dividir en n etapas y cada una se debe seccionar en aquellos circuitos que se necesitan para la microoperación. De esta manera, se puede simplificar el proceso de diseño considerablemente. Una vez que las diferentes piezas se diseñen separadamente, será posible cambiarlas para obtener una etapa típica del acumulador y luego cambiar las etapas en un acumulador completo.

Agregar B a A (p_1): La microoperación de suma se inicia cuando la variable de control p_1 es 1. Esta parte del acumulador puede usar un sumador en paralelo compuesto de circuitos sumadores completos como fue hecho con el ALU. El sumador completo en cada etapa i aceptará como entradas el estado presente de A_i, la entrada de datos B_i y el bit de arrastre previo C_i. El bit suma generado en el sumador completo debe ser trasferido al flip-flop A_i y el arrastre de salida C_{i+1} debe aplicarse al arrastre de entrada en la siguiente etapa.

La construcción interna de un circuito sumador completo puede ser simplificada si se considera que ésta opera como parte de un circuito secuencial. La tabla de estado de un sumador completo, cuando se considera como un circuito secuencial, se muestra en la Figura 9-18. El valor del flip-flop A_i anterior al pulso de reloj expecifica el estado presente en el cir-

cuito secuencial. El valor de A_i después de la aplicación de un pulso de reloj especifica el siguiente estado. El siguiente estado de A_i es una función de sus estados presentes y entradas B_i y C_i. El estado presente y las entradas en la tabla de estado corresponden a las entradas del sumador completo. El estado siguiente y la salida C_{i+1} corresponde a las salidas del sumador completo. Pero como éste es un circuito secuencial, A_i aparece en ambas columnas de estado presente y siguiente. El siguiente estado de A_i da el bit suma que debe trasferirse al flip-flop.

Las entradas de excitación para el flip-flop JK se listan en las columnas JA_i y KA_i. Estos valores se obtienen por el método enunciado en la Sección 6-7. Las funciones de entrada del flip-flop y las funciones de Boole para la salida se simplifican en los mapas de la Figura 9-18. La entrada J del flip-flop A_i designada como JA_i y la entrada K del flip-flop A_i designada como KA_i no incluyen la variable de control p_1. Estas dos ecuaciones deben afectar el flip-flop solamente cuando p_1 esté habilitada, por tanto deben aplicarse a una función AND con la variable de control p_1. La parte del circuito combinacional asociada con la microoperación de suma puede ser expresada con tres funciones de Boole:

$$JA_i = B_iC_i'p_1 + B_i'C_ip_1$$
$$KA_i = B_iC_i'p_1 + B_i'C_ip_1$$
$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$

Las primeras ecuaciones son idénticas y especifican una condición para complementar A_i. La tercera ecuación genera el arrastre de la siguiente etapa.

Estado presente	Entradas		Estado siguiente	Entradas de flip-flops		Salida
A_i	B_i	C_i	A_i	JA_i	KA_i	C_{i+1}
0	0	0	0	0	X	0
0	0	1	1	1	X	0
0	1	0	1	1	X	0
0	1	1	0	0	X	1
1	0	0	1	X	0	0
1	0	1	0	X	1	1
1	1	0	0	X	1	1
1	1	1	1	X	0	1

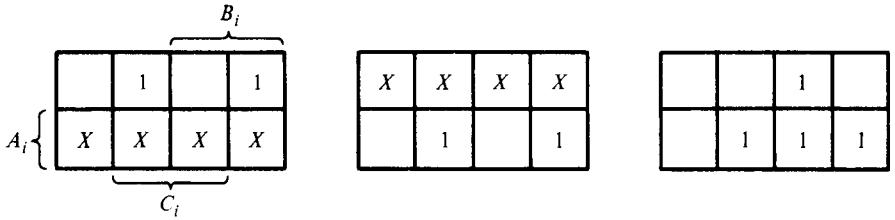


Figura 9-18 Tabla de excitación para la microoperación de suma

Borrar (p_2): La variable de control p_2 borra todos los flip-flops en el registro A. Para causar esta transición en el flip-flop JK se necesita solamente aplicar la variable de control p_2 a la entrada K del flip-flop. La entrada J se asumirá como 0 si nada se aplica a ella. Las funciones de entrada para la microoperación de borrado son:

$$JA_i = 0$$
$$KA_i = p_2$$

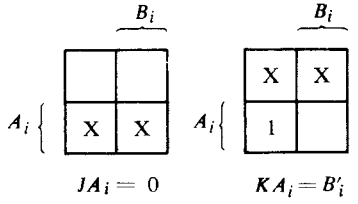
Complementar (p_3): La variable de control p_3 complementa el estado del registro A. Para causar esta transición en un flip-flop JK se necesita aplicar p_3 a ambas entradas J y K:

$$JA_i = p_3$$
$$KA_i = p_3$$

AND (p_4): La microoperación AND se inicia con la variable de control p_4. Esta operación forma la operación lógica AND entre A_i y B_i y

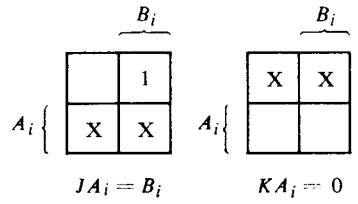
Estado presente	Entrada	Estado siguiente	Entradas de los flip-flops	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	0	0	X
1	0	0	X	1
1	1	1	X	0

(a) AND



Estado presente	Entrada	Estado siguiente	Entradas de los flip-flops	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	1	X	0

(b) OR



Estado presente	Entrada	Estado siguiente	Entradas de los flip-flops	
A_i	B_i	A_i	JA_i	KA_i
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1

(c) OR-exclusiva

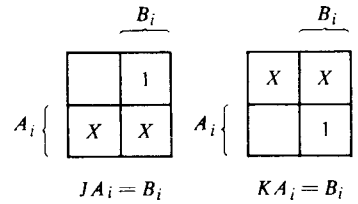


Figura 9-19 Tablas de excitación para las microoperaciones lógicas

trasfiere el resultado a A_i. La tabla de excitación para esta operación se da en la Figura 9-19(a). La siguiente etapa de A_i es 1 solamente cuando B_i y el estado presente de A_i sea igual a 1. Las funciones de entrada del flip-flop, las cuales se simplifican en los mapas, indican que la entrada K del flip-flop se habilita con el valor del complemento de B_i. Este resultado puede ser verificado de las condiciones listadas en la tabla de estado. Si $B_i = 1$ el estado presente y siguiente de A_i son iguales de manera que el flip-flop no tiene que sufrir un cambio de estado. Si $B_i = 0$, el siguiente estado de A_i debe ir a 0 y para lograrlo se habilita la entrada K del flip-flop. Las funciones de entrada para la microoperación AND deben incluir las variables de control que inician esta microoperación:

$$JA_i = 0$$
$$KA_i = B_i' p_4$$

OR (p_5): La variable de control p_5 inicia la operación lógica OR entre A_i y B_i con el resultado trasferido a A_i. La Figura 9-19(b) muestra la deducción de las funciones de entrada del flip-flop para esta operación. Las operaciones simplificadas en los mapas indican que la entrada J está habilitada cuando $B_i = 1$. Este resultado puede ser verificado a partir de la tabla de estado. Cuando $B_i = 0$, el estado presente y siguiente de A_i son iguales. Cuando $B_i = 1$, la entrada J se habilita y el estado siguiente de A_i se convierte en 1. Las funciones de entrada para las microoperaciones OR son:

$$JA_i = B_i p_5$$
$$KA_i = 0$$

OR-exclusiva (p_6): Esta operación forma la OR-exclusiva lógica entre A_i y B_i y trasfiere el resultado a A_i. La información pertinente para esta operación se muestra en la Figura 9-19(c). Las funciones de entrada de los flip-flops son:

$$JA_i = B_i p_6$$
$$KA_i = B_i p_6$$

Desplazamiento a la derecha (p_7): Esta operación desplaza el contenido del registro A una posición a la derecha. Esto significa que el valor del flip-flop A_{i+1}, el cual está una posición a la izquierda de la etapa i, debe ser trasferido al flip-flop A_i. Esta trasferencia se expresa por medio de las funciones de entrada:

$$JA_i = A_{i+1} p_7$$
$$KA_i = A_{i+1}' p_7$$

Desplazamiento a la izquierda (p_8): Esta operación desplaza el registro A una posición a la izquierda. Para este caso el valor de A_{i-1}, el

cual está una posición a la derecha de la etapa i, debe ser trasferido a A_i. Esta trasferencia se expresa por medio de las funciones de entrada:

$$JA_i = A_{i-1}p_8$$
$$KA_i = A'_{i-1}p_8$$

Incremento (p_9): Esta operación incrementa el contenido del registro A en uno; en otras palabras, el registro se comporta como un contador binario asincrónico con p_9 habilitando la cuenta. Un contador sincrónico de 3 bits se muestra en la Figura 9-20. Este es similar al contador de la Figura 7-17 de la Sección 7-5 donde se discute en detalle la operación de los contadores binarios sincrónicos. Del diagrama, se observa que cada etapa se complementa cuando un arrastre de entrada $E_i = 1$. Cada etapa genera también un arrastre de salida E_{i+1} para la siguiente etapa de la izquierda. La primera etapa es una excepción ya que ésta se complementa con el habilitador de cuenta p_9. Las funciones de Boole de una etapa típica pueden ser expresadas de la siguiente manera:

$$JA_i = E_i$$
$$KA_i = E_i$$
$$E_{i+1} = E_iA_i \qquad i = 1, 2, \ldots, n$$
$$E_1 = p_9$$

El arrastre de entrada E_i a la etapa, se usa para complementar el flip-flop A_i. Cada etapa genera un arrastre para la siguiente, aplicando la función

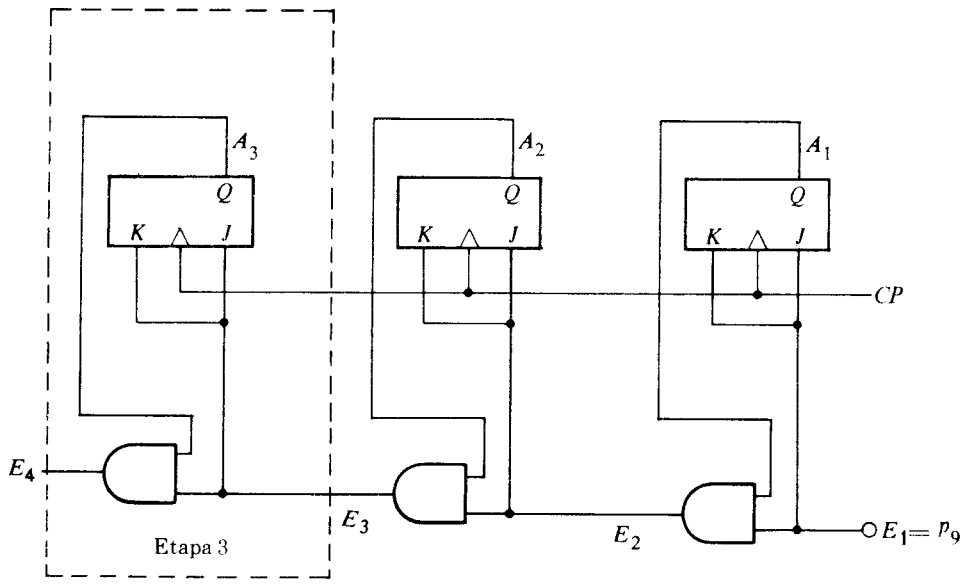


Figura 9-20 Contador binario sincrónico de 3 bits

AND al arrastre de entrada y a A_i. El arrastre de entrada que va a la primera etapa es E_1 y debe ser igual a la variable de control p_9 la cual habilita la cuenta.

Comprobación del cero (Z): La variable Z es una salida del acumulador usado para indicar un contenido de cero en el registro A. Esta salida es igual al binario 1 cuando todos los flip-flops se hayan borrado. Cuando un flip-flop se borra, su salida complementada Q' es igual a 1. La Figura 9-21 muestra las primeras tres etapas del acumulador para comprobar un contenido de ceros. Cada etapa genera una variable z_{i+1} al aplicar la función AND a la salida complementada de A_i y a una variable de entrada z_i. De esta manera una cadena de compuertas AND a lo largo de todas las etapas indicará si todos los flip-flops están borrados. Las funciones de Boole para una etapa típica pueden ser expresadas de la siguiente manera:

$$z_{i+1} = z_i A_i' \qquad i = 1, 2, \ldots, n$$
$$z_1 = 1$$
$$z_{n+1} = Z$$

La variable Z se convierte en 1 si la señal de salida de la última etapa z_{n+1} es 1.

Una etapa del acumulador

Una etapa típica del acumulador consiste de todos los circuitos que fueron deducidos para las microoperaciones individuales. Las variables de control p_1 hasta p_9 son excluyentes mutuamente de manera que los circuitos lógicos correspondientes pueden ser combinados con una operación OR. Combinando todas las funciones de entrada para las entradas J y K del flip-flop

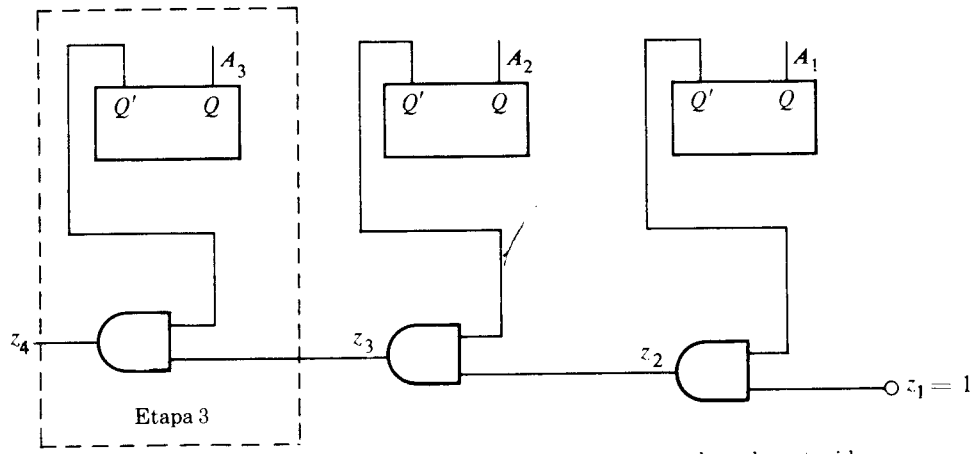


Figura 9-21 Cadena de compuertas AND para comprobar el contenido de ceros en el registro

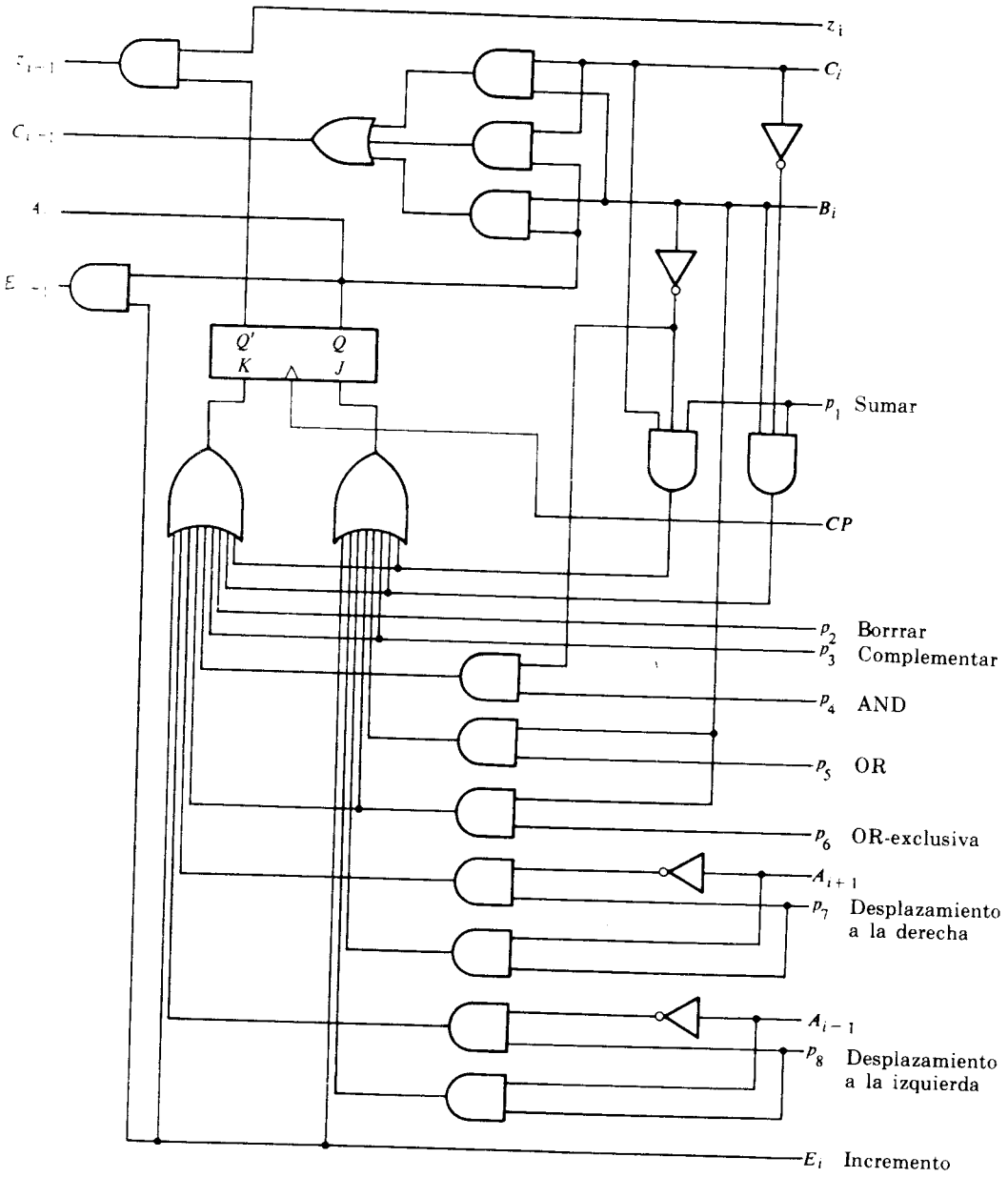


Figura 9-22 Una etapa típica del acumulador

A_i se produce un conjunto compuesto de funciones de entrada de Boole para un estado típico:

$$JA_i = B_iC_i'p_1 + B_i'C_ip_1 + p_3 + B_ip_5 + B_ip_6 + A_{i+1}p_7 + A_{i-1}p_8 + E_i$$
$$KA_i = B_iC_i'p_1 + B_i'C_ip_1 + p_2 + p_3 + B_i'p_4 + B_ip_6 + A_{i+1}'p_7 + A_{i-1}'p_8 + E_i$$

Cada etapa en el acumulador debe generar también los arrastres para la siguiente etapa:

$$C_{i+1} = A_iB_i + A_iC_i + B_iC_i$$
$$E_{i+1} = E_iA_i$$
$$z_{i+1} = z_iA_i'$$

El diagrama lógico de una etapa típica de un acumulador se muestra en la Figura 9-22. Esta es la configuración directa de las funciones de Boole listadas anteriormente. El diagrama es un circuito compuesto que incluye los circuitos individuales asociados con cada microoperación. Los diferentes circuitos se combinan con dos compuertas OR en las entradas J y K del flip-flop A_i.

Cada etapa del acumulador tiene ocho entradas de control p_1 hasta p_8 que indican una de las ocho microoperaciones posibles. La variable de control p_9 se aplica solamente a la primera etapa para habilitar la operación de incremento a través de la entrada E_i, hay otras seis entradas en el circuito. B_i es el bit de datos de los terminales B que suministran las entradas al acumulador. C_i es el arrastre de entrada de la etapa previa a la derecha. A_{i-1} viene de la salida del flip-flop que está una posición a la derecha y A_{i+1} viene del flip-flop que está una posición a la izquierda. E_i es el arrastre de entrada para la operación de incremento y z_i se usa para formar la cadena para detección de cero. El circuito tiene cuatro salidas: A_i es la salida del flip-flop, C_{i+1} es el arrastre para la siguiente etapa, E_{i+1} es el arrastre de incremento para la siguiente etapa y z_{i+1} es para la siguiente etapa a la izquierda, para formar la cadena para la detección de cero.

Acumulador completo

Un acumulador completo con n bits requiere n estados conectados en cascada con cada etapa conteniendo el circuito mostrado en la Figura 9-22. Todas las variables de control excepto p_9 deben ser aplicadas a cada etapa. Las otras entradas y salidas en cada etapa deben estar conectadas en cascada para formar un acumulador completo.

La interconexión entre etapas para formar un acumulador completo se ilustra en el acumulador de 4 bits mostrado en la Figura 9-23. Cada bloque en el diagrama representa el circuito de la Figura 9-22. El número de la parte superior de cada bloque representa la posición de bit en el acumulador. Todos los bloques reciben ocho variables de control p_1 hasta p_8 y los pulsos de reloj del CP. Las otras seis entradas y cuatro salidas en cada

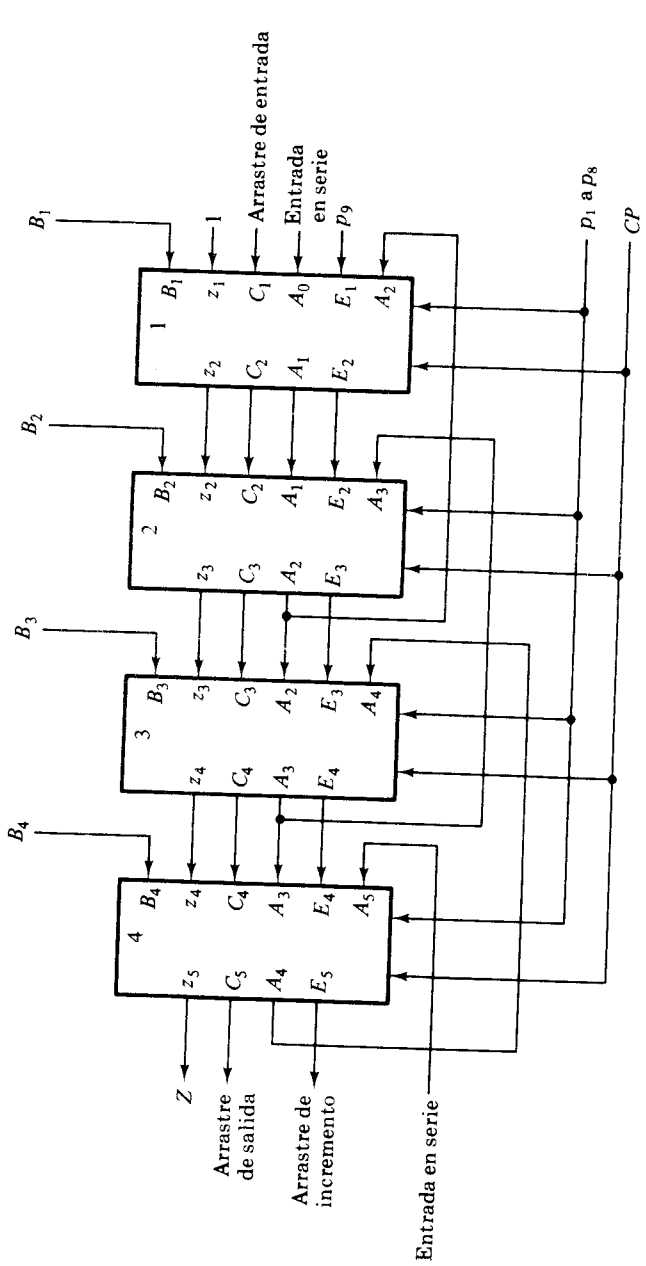


Figura 9-23 Acumulador de 4 bits construido con cuatro etapas

bloque son idénticas a aquellas de una etapa típica, excepto que el suscrito i se remplaza por el número particular en cada bloque.

El circuito tiene cuatro entradas B. La cadena de detección de cero se obtiene conectando las variables z en cascada, con el primer bloque que recibe una constante binaria 1. La última etapa en esta cadena produce la variable de detección de cero Z. Los arrastres de la suma aritmética se conectan en cascada como en los circuitos del sumador completo. La entrada en serie para la operación de desplazamiento a la izquierda, va a la entrada A_0 la cual corresponde a A_{i-1} en la primera etapa. La entrada en serie para la operación de desplazamiento a la derecha, va a la entrada A_5 la cual corresponde a A_{i+1} en la cuarta y última etapa. La operación de incremento se habilita con la variable de control p_9 en la primera etapa. Los otros bloques reciben el incremento de arrastre de la etapa previa.

El número total de terminales en un acumulador de 4 bits es 25 incluyendo los terminales para las salidas A. Agregando dos terminales más para la fuente de poder el circuito se puede encapsular dentro de un CI que tenga 27 ó 28 patillas. El número de terminales para las variables de control pueden reducirse de 9 a 4 si se agrega un decodificador en el CI. En tal caso, la cuenta de las patillas del CI pueden ser reducidas a 22 y el acumulador puede ser extendido a 16 microoperaciones sin agregar patillas externas.

REFERENCIAS

1. Mano, M. M., *Computer System Architecture.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976.
2. *The TTL Data Book for Design Engineers.* Dallas, Texas: Texas Instruments, Inc., 1976.
3. *The Am2900 Bipolar Microprocessor Family Data Book.* Sunnyvale, Calif.: Advanced Micro Devices, Inc., 1976.
4. Sobel, H. S., *Introduction to Digital Computer Design.* Reading, Mass.: Addison-Wesley Publishing Co., 1970.
5. Kline, R. M., *Digital Computer Design.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977.
6. Chirlian, P. M., *Analysis and Design of Digital Circuits and Computer Systems.* Champaign, Ill.: Matrix Publishing, Inc., 1976.

PROBLEMAS

9-1. Modifique la unidad procesadora de la Figura 9-1 de manera que el registro de destino seleccionado sea siempre el mismo registro que se selecciona con el bus A. ¿Cómo afecta esto al número de multiplexores y al número de líneas de selección usados?

9-2. Un procesador con un bus organizado como en la Figura 9-1 consiste de 15 registros. ¿Cuántas líneas de selección hay en cada multiplexor y en el decodificador de destino?

9-3. Asuma que cada registro en la Figura 9-1 es de 8 bits de largo. Dibuje un diagrama de bloque detallado para el recuadro marcado MUX que selecciona el

registro para el bus A. Muestre que la selección puede hacerse con ocho multiplexores de 4 a 1 línea.

9-4. Una unidad procesadora emplea una memoria tapón como en la Figura 9-2. El procesador consiste de 64 registros de ocho bits cada uno.

(a) ¿Cuál es el tamaño de la memoria tapón? (scratchpad memory)
(b) ¿Cuántas líneas se necesitan para la dirección?
(c) ¿Cuántas líneas hay para los datos de entrada?
(d) ¿Cuál es el tamaño del MUX que selecciona entre la entrada de datos y la salida del registro de desplazamiento?

9-5. Muestre un diagrama de bloque detallado para la unidad procesadora de la Figura 9-4 cuando las entradas B vienen de:

(a) Ocho registros procesadores que forman un sistema de bus.
(b) Una unidad de memoria con direcciones y registros separadores.

9-6. El ALU de 4 bits de la Figura 9-5 se incluye dentro de un CI. Muestre las conexiones entre tres CI para conformar un ALU de 12 bits. Asigne los arrastres de entrada y salida en el ALU de 12 bits.

9-7. El CI TTL tipo 7487 es un elemento de 4 bits verdadero/complemento, cero/uno. Una etapa de este CI se muestra en la Figura P9-7.

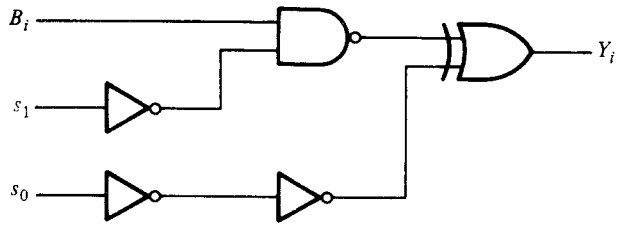


Figura P9-7 Circuito verdadero/complemento, uno/cero

(a) Deduzca las funciones de Boole para las salidas Y_i como función de las entradas B_i, s_1 y s_0.
(b) Dibuje la tabla de verdad para el circuito.
(c) Dibuje una tabla de función (similar a aquella mostrada en la Figura 9-7) y verifique la operación del circuito.

9-8. Modifique el circuito aritmético de la Figura 9-8 incluyendo una tercera variable de selección s_2. Cuando $s_2 = 1$ el circuito modificado es idéntico al circuito original. Cuando $s_2 = 0$, todas las entradas A a los sumadores completos son inhibidas y se colocan ceros de remplazo.

(a) Dibuje el diagrama lógico de una etapa del circuito modificado.
(b) Haga un análisis similar a aquel de la Figura 9-6 para determinar las ocho operaciones cuando $s_2 = 0$.
(c) Liste las nuevas funciones de salida en forma de tabla.

9-9. Determine las operaciones aritméticas obtenidas en ocho bloques de la Figura 9-6, si en cada caso la entrada A se cambia a $\overline{A}$ (complemento de A).

9-10. Diseñe un circuito aritmético con una variable de selección s y dos entradas de datos A y B. Cuando $s = 0$, el circuito realiza la operación de suma $F = A + B$. Cuando $s = 1$, el circuito realiza la operación de incremento $F = A + 1$.

9-11. La sustracción binaria directa $F = A - B$ produce una diferencia correcta si $A \geq B$. ¿Cuál podría ser el resultado si $A < B$? Determine la relación entre el resultado obtenido en F y el bit de préstamo en la posición más significativa.

9-12. Diseñe un circuito aritmético con dos variables de selección s_1 y s_0 que genera las siguientes operaciones aritméticas. Dibuje el diagrama lógico de una etapa típica.

s_1	s_0	$C_{in} = 0$	$C_{in} = 1$
0	0	$F = A + B$	$F = A + B + 1$
0	1	$F = A$	$F = A + 1$
1	0	$F = \bar{B}$	$F = \bar{B} + 1$
1	1	$F = A + \bar{B}$	$F = A + \bar{B} + 1$

9-13. Diseñe un circuito aritmético con dos variables de selección s_1 y s_0 que genere las siguientes operaciones aritméticas. Dibuje el diagrama lógico de una etapa típica.

s_1	s_0	$C_{in} = 0$	$C_{in} = 1$
0	0	$F = A$	$F = A + 1$
0	1	$F = A - B - 1$	$F = A - B$
1	0	$F = B - A - 1$	$F = B - A$
1	1	$F = A + B$	$F = A + B + 1$

9-14. Las siguientes relaciones de la operación OR-exclusiva fueron usadas para derivar las operaciones lógicas de la Tabla 9-3.

(a) $x \oplus 0 = x$

(b) $x \oplus 1 = x'$

(c) $x \oplus y' = x \odot y$

Pruebe que estas relaciones son válidas.

9-15. Deduzca un circuito combinacional mínimo que genere todas las 16 funciones lógicas listadas en la Tabla 2-5. Use cuatro variables de selección. *Sugerencia:* Use un multiplexor de 4×1 invertido, es decir, use las entradas normales del multiplexor como las líneas de selección para la unidad lógica.

9-16. Modifique el circuito aritmético de la Figura 9-8 en un ALU con la variable de selección de modo s_2. Cuando $s_2 = 0$, el ALU es idéntico al circuito aritmético. Cuando $s_2 = 1$, el ALU genera las funciones lógicas de acuerdo a la siguiente tabla:

s_2	s_1	s_0	Salida	Función
1	0	0	$F = A \wedge B$	AND
1	0	1	$F = A \oplus B$	XOR
1	1	0	$F = A \vee B$	OR
1	1	1	$F = \bar{A}$	NOT

9-17. Una unidad lógica aritmética es similar a la mostrada en la Figura 9-13, excepto que las entradas a cada circuito sumador completo están de acuerdo a las siguientes funciones de Boole:

$$X_i = A_iB_i + (s_2s_1's_0')'A_i + s_2s_1s_0'B_i$$
$$Y_i = s_0B_i + s_1B_i'(s_2s_1s_0')'$$
$$Z_1 = s_2'C_i$$

Determine las 12 funciones del ALU.

9-18. La operación realizada en un ALU es $F = A + \overline{B}$ (A más el complemento de 1 de B).

(a) Determine el valor de salida F cuando $A = B$. Permita que esta condición ponga a uno un bit de condición E.

(b) Determine que la condición $C_{out} = 1$. Permita que esta condición ponga a uno el bit de condición C.

(c) Deduzca una tabla para las seis relaciones listadas en la Tabla 9-5 en términos de las condiciones de los bits de condición E y C definidos anteriormente.

9-19. Una unidad de proceso tiene un registro de condición de diez bits, uno para cada una de las condiciones listadas en las Tablas 9-5 y 9-6. (Las condiciones iguales y desiguales son comunes a ambas tablas.) Dibuje un diagrama lógico mostrando las compuertas que van de las salidas del ALU a los diez bits del registro de condición.

9-20. Dos números con signo se agregan en un ALU y su suma se trasfiere al registro R. Los bits de condición S (signo) y V (sobrecapacidad) se afectan durante la trasferencia. Pruebe que la suma puede dividirse ahora por 2 de acuerdo a la proposición:

$$R \leftarrow \text{shr } R, \qquad R_n \leftarrow S \oplus V$$

donde R_n es el bit de signo (posición extrema izquierda) del registro R.

9-21. Agregue otro multiplexor al registro de desplazamiento de la Figura 9-15 con dos líneas de selección separadas G_1 y G_0. Este multiplexor se usa para especificar la entrada en serie I_R durante la operación de desplazamiento a la derecha de la siguiente manera:

G_1	G_0	Función
0	0	Colocar ceros a I_R
0	1	Hacer un desplazamiento circular
1	0	Hacer un desplazamiento circular con arrastre
1	1	Coloque el valor de $S \oplus V$ para el desplazamiento aritmético (ver Problema 9-20)

Muestre la conexión del multiplexor entre el registro de condición y el desplazamiento.

9-22. El selector de desplazamiento H definido para el procesador de la Figura 9-16 tiene tres variables H_2, H_1 y H_0. Las dos últimas variables de selección se

usan para el registro de desplazamiento especificado en la Tabla 9-7. Diseñe el circuito asociado con la variable de selección H_2.

9-23. Especifique la palabra de control que puede ser aplicada al procesador de la Figura 9-16 para configurar las siguientes microoperaciones:

(a) $R2 \leftarrow R1 + 1$
(b) $R3 \leftarrow R4 + R5$
(c) $R6 \leftarrow \overline{R6}$
(d) $R7 \leftarrow R7 - 1$
(e) $R1 \leftarrow \text{shr } R1$
(f) $R2 \leftarrow \text{clc } R2$
(g) $R3 \leftarrow R4 \oplus R5$
(h) $R6 \leftarrow R7$

9-24. Es necesario calcular el valor promedio de cuatro números binarios sin signo almacenados en los registros $R1$, $R2$, $R3$ y $R4$ del procesador definido en la Figura 9-16. El valor promedio se debe almacenar en el registro $R5$. Los otros dos registros en el procesador pueden usarse para resultados intermedios. Se debe tener cuidado de no causar sobrecapacidad.

(a) Dé la lista de la secuencia de microoperaciones en forma simbólica.
(b) Liste las palabras de control binarias correspondientes.

9-25. La siguiente secuencia de microoperaciones se realiza en el acumulador definido en la Sección 9-10.

$$p_3: \quad A \leftarrow \overline{A}$$
$$p_9: \quad A \leftarrow A + 1$$
$$p_1: \quad A \leftarrow A + B$$
$$p_3: \quad A \leftarrow \overline{A}$$
$$p_9: \quad A \leftarrow A + 1$$

(a) Determine el contenido de A después de cada microoperación si inicialmente $A = 1101$ y la entrada $B = 0110$.
(b) Repita (a) con las condiciones iniciales $A = 0110$ y $B = 1101$.
(c) Repita (a) con las condiciones iniciales $A = 0110$ y $B = 0110$.
(d) Pruebe que la anterior secuencia de microoperaciones realiza $(A - B)$ si $A \geq B$, o el complemento de 2 de $(B - A)$ si $A < B$.

9-26. Usando flip-flops JK diseñe una etapa típica de un registro A que realiza la microoperación de sustracción:

$$p_{10}: \quad A \leftarrow A - B$$

Use los circuitos sumadores completos (Sección 4-4) con bits de préstamo de entrada y salida K_i y K_{i+1}.

9-27. Usando flip-flops JK diseñe una etapa típica de un registro que realice las siguientes microoperaciones lógicas:

p_{11}: $A \leftarrow \overline{A \vee B}$ NOR
p_{12}: $A \leftarrow \overline{A \wedge B}$ NAND
p_{13}: $A \leftarrow A \odot B$ Equivalencia

9-28. Deduzca las operaciones de Boole de una etapa típica de una microoperación de decremento:

$$p_{14}: \quad A \leftarrow A - 1$$

9-29. Usando flip-flops tipo T diseñe un registro de 4 bits que ejecute la microoperación de complemento de 2:

$$P: \quad A \leftarrow \bar{A} + 1$$

Del resultado obtenido, muestre que una etapa típica puede ser expresada por las siguientes funciones de Boole:

$$TA_i = PE_i \qquad i = 1, 2, 3, \ldots, n$$
$$E_{i+1} = A_i + E_i$$
$$E_1 = 0$$

9-30. Un acumulador de 4 bits realiza 15 microoperaciones con variables de control p_1 a p_{15}. El circuito se encapsula en un CI con sólo cuatro terminales disponibles para seleccionar la microoperación. Diseñe un circuito (dentro del CI) que se deba agregar entre los cuatro terminales y las 15 variables de control. Incluya una condición de no operación.

Diseño de lógica de control

10

10-1 INTRODUCCION

El proceso del diseño lógico es una tarea compleja. Muchas instalaciones desarrollan varias técnicas de diseño de computador automatizado para facilitar el proceso de diseño. Sin embargo las especificaciones para el sistema y el desarrollo de procedimientos algorítmicos para lograr las tareas requeridas de procesamiento de datos no pueden ser automatizados y requieren un razonamiento mental del diseñador humano.

La parte de mayor desafío y creatividad del diseño es el establecimiento de objetivos de diseño y la formulación de algoritmos y procesamientos para lograr los objetivos enunciados. Esta tarea requiere una cantidad considerable de experiencia e ingenuidad por parte del diseñador. *Un algoritmo* es un procedimiento para obtener una solución al problema. Un *algoritmo diseñado* es un procedimiento para configurar el problema con una pieza dada de equipo. El desarrollo del algoritmo diseñado no puede comenzar hasta que el diseñador esté seguro de dos cosas. Primero, el problema entre manos debe comprenderse completamente. Segundo, se debe asumir una configuración inicial del equipo para conformar el procedimiento. A partir del enunciado del problema y de la disponibilidad de equipo se busca una solución y se forma un algoritmo. El algoritmo se enuncia mediante un número finito de pasos de procedimientos bien definidos.

La información binaria encontrada en un sistema digital se almacena en un procesador o registros de memoria y puede ser constituida por datos o información de control. Los datos son elementos discretos de información que se manipulan por microoperaciones. La información de control suministra señales de mandos para especificar la secuencia de microoperaciones. La lógica de diseño de un sistema digital es un proceso para deducir los circuitos digitales que realizan datos de procesamientos y de circuitos digitales que suministran señales de control.

La temporización de todos los registros en un sistema digital sincrónico se controla por medio de un generador de pulsos de reloj maestros. Los pulsos de reloj se aplican a todos los flip-flops y los registros en el sistema, incluyendo los flip-flops y registros en la unidad de control. Los pulsos continuos de reloj no cambian el estado de un registro a no ser que el registro se habilite por la señal de control. Las variables binarias, que controlan

las variables de selección y las entradas de habilitación de los registros, se generan en la unidad de control. Las salidas de la unidad de control seleccionan y habilitan la parte del procesador de datos del sistema y también determinan el siguiente estado de la unidad de control en sí misma.

La relación entre la unidad de control y el procesador de datos en un sistema digital se muestra en la Figura 10-1. La parte del procesador de datos puede ser una unidad procesadora de propósito general, o puede consistir de registros individuales y funciones digitales asociadas. El control inicia todas las microoperaciones en el procesamiento de datos. La lógica de control que genera las señales para dar secuencia a las microoperaciones en un circuito secuencial cuyos estados internos indican las funciones de control del sistema. En un tiempo dado, el estado de control secuencial inicia un conjunto de microoperaciones preseleccionadas. El control secuencial pasa el siguiente estado o inicia otras microoperaciones dependiendo de las condiciones presentes y otras entradas. Así, el circuito digital que actúa como la lógica de control suministra una secuencia de tiempo de señales para iniciar las microoperaciones en la parte del procesador de datos del sistema.

El diseño de un sistema digital que requiere una secuencia de control comienza con la suposición de la disponibilidad de variables de tiempo. Se diseña cada variable en la secuencia por medio de un estado y luego se forma un diagrama de estado o una representación equivalente para la transición entre estados. Paralelamente con el desarrollo de la secuencia de control se hace una lista de microoperaciones que se van a iniciar, para cada estado de control. Si el sistema es muy complicado para un diagrama de estado, puede ser conveniente especificar enteramente el sistema por el método de trasferencia entre registros por medio de las funciones de control y las proposiciones de microoperaciones.

La secuencia de control y las relaciones de trasferencia entre registros pueden deducirse directamente de la especificación en palabras del problema. Sin embargo es conveniente algunas veces usar una representación intermedia para describir la secuencia necesaria de operaciones del sistema. Dos representaciones, útiles en el diseño de sistemas que necesitan control, son los diagramas de tiempo y los flujogramas.

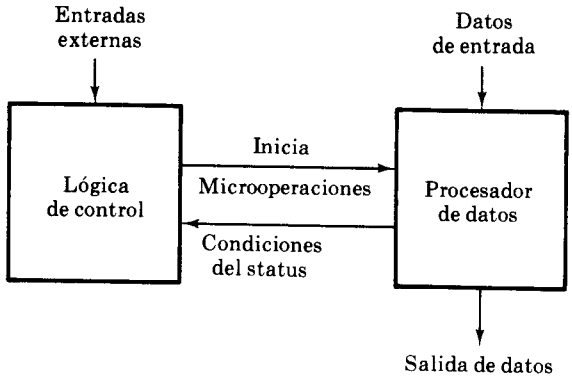


Figura 10-1 Interacción entre el control y el procesador de datos

Un *diagrama de tiempo* clarifica la secuencia de tiempo y otras relaciones entre las diferentes señales de control del sistema. En un circuito secuencial con reloj, los pulsos de reloj sincronizan todas las operaciones incluyendo las señales de transición en las variables de control. En un sistema asincrónico una señal de transición en una variable de control puede causar un cambio a otra variable de control. Un diagrama de tiempo es muy útil en un control asincrónico ya que provee una representación ilustrativa de los cambios requeridos y las transiciones de todas las variables de control.

Un *flujograma* es una manera conveniente de especificar la secuencia de pasos de procedimiento y formas de decisión para un algoritmo. Un flujograma para un algoritmo diseñado usaría normalmente los nombres de las variables de los registros definidos en la configuración inicial del equipo. Este traslada un algoritmo de su enunciado en palabras a un diagrama de flujo de información que enumera la secuencia de operaciones de trasferencia entre registros conjuntamente con las condiciones necesarias para su ejecución.

Un flujograma es un diagrama que consiste de bloques conectados por medio de líneas directas. Dentro de los bloques se especifican los pasos procedimentales para configurar el algoritmo. Las líneas directas entre bloques indican el camino que se va a tomar de un paso procedimental al siguiente. Se usan dos tipos mayores de bloques: un bloque rectangular indica un bloque de *función* dentro del cual se listan las microoperaciones. Un bloque en forma de diamante es un bloque de *decisión* dentro del cual se lista una condición actual dada. Un bloque de decisión tiene dos o más caminos alternos y el camino que se toma depende del valor de la condición de estado especificada dentro del bloque.

Un flujograma es muy similar a un diagrama de estado. Cada bloque de función en el flujograma es equivalente a un estado en un diagrama de estado. El bloque de decisión en el flujograma es equivalente a la información binaria escrita por conducto de las líneas dirigidas que conectan dos estados en un diagrama de estado. Como consecuencia, es conveniente algunas veces expresar un algoritmo por medio de un flujograma del cual se puede deducir el diagrama de estado de control.

En este capítulo se presentan cuatro configuraciones posibles para una unidad de control. Las diferentes configuraciones se presentan en forma de diagrama de bloque para darle énfasis a las diferencias en organización. Se demuestran entonces varios procedimientos disponibles para el diseño de control lógico analizando ejemplos específicos.

El diseño de la lógica de control no puede separarse del desarrollo del algoritmo para resolver un problema de diseño. Sin embargo, la lógica de control se relaciona directamente a la parte del procesador de datos del sistema que éste controla. Como consecuencia, los ejemplos presentados en este capítulo comienzan con el desarrollo de un algoritmo para configurar el problema dado. La parte del procesamiento de datos del sistema se deduce entonces del algoritmo enunciado. Solamente hasta que se haga lo anterior se puede proceder a mostrar el diseño del control que da secuencia al procesador de datos de acuerdo a los pasos especificados por el algoritmo.

10-2 ORGANIZACION DEL CONTROL

Una vez que se haya establecido la secuencia de control se puede diseñar el sistema secuencial que configura las operaciones de control. Como el control es un circuito secuencial, éste se puede diseñar por el procedimiento lógico secuencial enunciado en el Capítulo 6. Sin embargo, este método es poco práctico en la mayoría de los casos debido al gran número de estados que el circuito de control puede tener. Los métodos de diseño que usan estados y tablas de excitación pueden usarse en teoría, pero en la práctica son engorrosos y difíciles de manejar. Además, los circuitos de control obtenidos por este método requieren por lo general un número excesivo de flip-flops y compuertas, lo cual implica el uso de compuertas SSI. Este tipo de configuración es ineficiente con respecto al número de CI que se usan y al número de alambres que deben ser interconectados. El principal objetivo del diseño de lógica de control debe ser el desarrollo de un circuito que configure la secuencia de control deseada de una manera lógica y directa. El esfuerzo de minimizar el número de circuitos tendería a producir una configuración irregular, lo cual haría difícil para cualquier persona diferente al diseñador, el reconocimiento de la secuencia de eventos por los cuales pasa el control. Como consecuencia podría ser difícil dar servicio y mantener el equipo cuando está en operación.

Debido a las razones citadas anteriormente los diseñadores con experiencia lógica usan métodos para el diseño de lógica de control que pueden ser considerados como una extensión del método lógico secuencial clásico combinado con el método de trasferencia entre registros. En esta sección se consideran cuatro métodos de organización de control.

1. Método de un flip-flop por estado.
2. Método del registro de secuencia y el decodificador.
3. Control PLA.
4. Control del microprograma.

Los primeros dos métodos resultan en un circuito que debe usar circuitos SSI y MSI para la configuración. Los diferentes circuitos se interconectan con alambres para formar una red circuito de control. Una unidad de control configurada con elementos SSI y MSI se denota como un control a base de materiales interconectados. Si se necesitan alteraciones o modificaciones, los circuitos se deben alambrar de nuevo para cumplir con las nuevas especificaciones. Esto es en contraste al PLA o control de microprograma el cual usa un elemento LSI tal como un arreglo lógico programable o una memoria de solamente lectura. Cualquier alteración o modificación en el microprograma de control puede lograrse fácilmente sin cambiar de alambrado removiendo la ROM de su base y colocando otra ROM programada para copar las nuevas especificaciones.

Se explica ahora en términos generales cada método. Las secciones subsiguientes de este capítulo tratan con ejemplos específicos que demuestran el diseño detallado de las unidades de control de los cuatro métodos.

Método de un flip-flop por estado

Este método usa un flip-flop por estado en el circuito secuencial de control. Solamente se pone a uno un flip-flop en un tiempo dado, los demás se ponen a cero. Se hace programar un solo bit de un flip-flop a otro bajo el control de la lógica de decisión. En tal arreglo cada flip-flop representa un estado y se activa solamente cuando el bit de control se trasfiere a éste.

Es obvio que este método no usa un número mínimo de flip-flops para el circuito secuencial. De hecho, éste usa un número máximo de flip-flops. Por ejemplo un circuito secuencial con 12 estados requiere un mínimo de cuatro flip-flops porque $2^3 < 12 < 2^4$. Aun por medio de este método el circuito de control usa 12 flip-flops para cada estado.

La ventaja de un flip-flop por método de estado es la simplicidad con la cual se diseña. Este tipo de controlador puede diseñarse por inspección a partir de un diagrama de estado que describe la secuencia de control. A primera vista, parece que este método aumentará el costo del sistema ya que se necesita un mayor número de flip-flops, pero, este método ofrece otras ventajas que no son aparentes a primera vista. Por ejemplo, éste ofrece un ahorro de esfuerzos en el diseño, un aumento en la simplicidad operacional y una disminución potencial en los circuitos combinacionales requeridos para configurar el circuito secuencial completo.

La Figura 10-2 muestra la configuración de una lógica de control secuencial de cuatro estados, que usa cuatro flip-flops tipo D: un flip-flop por estado T_i, $i = 0, 1, 2, 3$. En cualquier intervalo de tiempo dado entre dos pulsos de reloj solamente un flip-flop es igual a 1, el resto será igual a 0. La transición del estado presente al siguiente es una función del presente T_i que es 1 y de ciertas condiciones de entrada. El siguiente estado se manifiesta cuando el flip-flop anterior se borra y el nuevo se pone a uno. Cada una de las salidas del flip-flop se conecta a la sección de procesamiento de datos del sistema digital para iniciar ciertas microoperaciones. Las otras salidas de control mostradas en el diagrama son una función de las T y de las entradas externas. Estas salidas pueden también iniciar microoperaciones.

Si el circuito de control no necesita entradas externas para su cadencia, el circuito se reduce a un circuito de desplazamiento simple con un solo bit que se desplaza de una posición a la siguiente. Si la secuencia de control debe repetirse una y otra vez, el control se reduce a un contador de anillo. *Un contador de anillo* es un registro de desplazamiento con la salida del último flip-flop conectado a la entrada del primer flip-flop. En un contador de anillo el solo bit se desplaza continuamente de una posición a la siguiente de una manera circular. Por esta razón el método de un flip-flop por estado se llama algunas veces un *controlador del contador de anillo.*

Registro de secuencia y método del decodificador

Este método usa un registro para darle secuencia a los estados de control. El registro se decodifica para suministrar una salida por cada estado. El circuito tendrá 2^n estados y el decodificador 2^n salidas, p ra n flip-flops

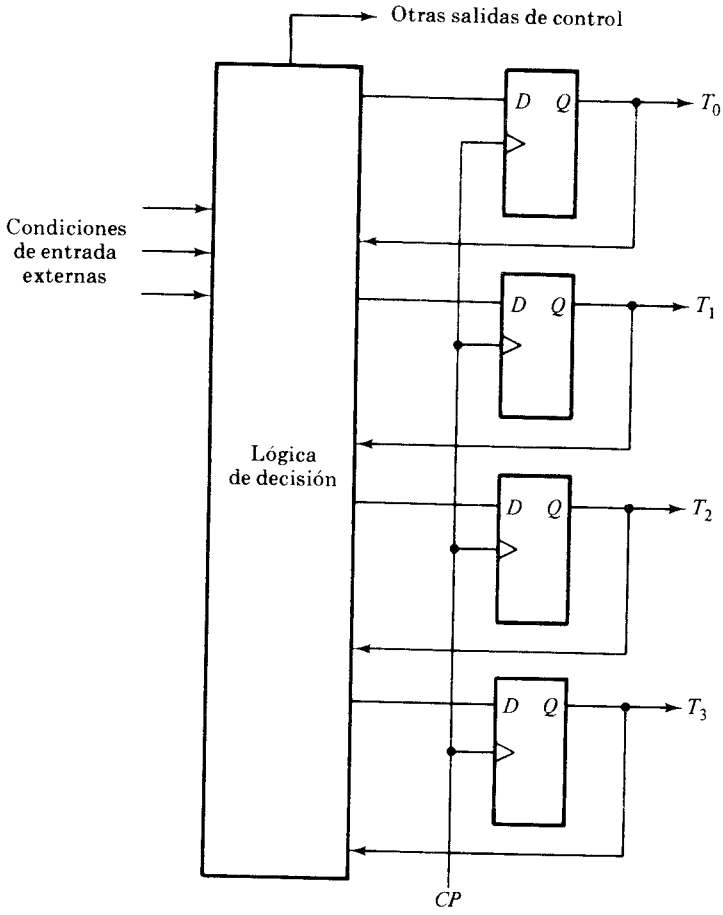


Figura 10-2 Lógica de control con un flip-flop por estado

en el registro de secuencia. Por ejemplo, un registro de 4 bits puede estar en cualquiera de los 16 estados. Un decodificador de 4×16 tendrá 16 salidas, una para cada estado del registro. Tanto el registro de secuencia como el decodificador son componentes MSI.

La Figura 10-3 muestra la configuración de una lógica de control secuencial de cuatro estados. El registro de secuencia tiene dos flip-flops y el decodificador establece salidas separadas para cada estado en el registro. La transición al siguiente estado en el registro de secuencia es una función del estado presente y de las condiciones de entrada externas. Como las salidas del decodificador están de alguna forma disponibles, es conveniente usarlas como variables de estado presente en vez de usar directamente las salidas de los flip-flops. Otras salidas que son función del estado presente y de las entradas externas pueden iniciar microoperaciones en adición a las salidas del decodificador.

Si el circuito de control de la Figura 10-3 no necesita entradas externas, el registro de secuencia se reduce a un contador que continuamente

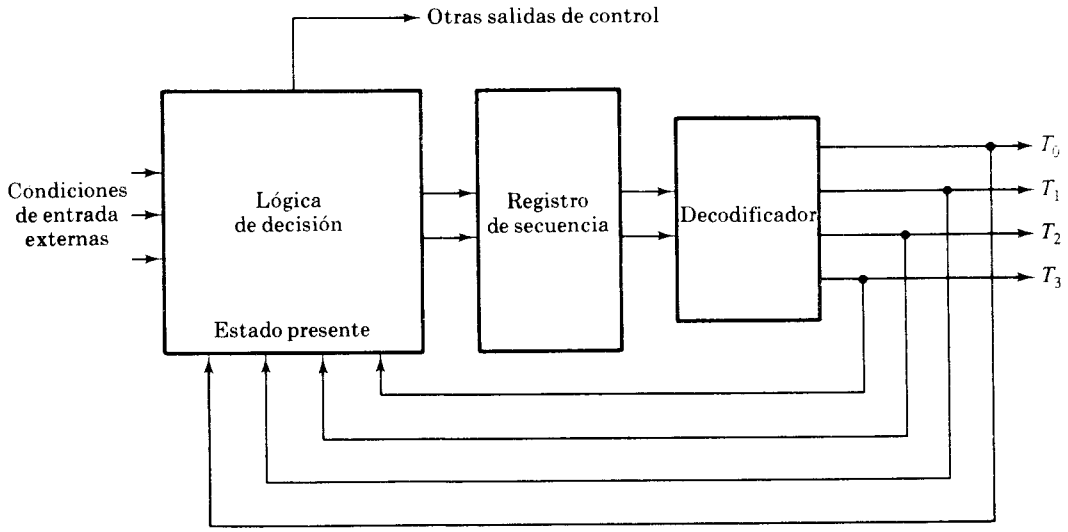


Figura 10-3 Lógica de control con registro de secuencia y decodificador

hace secuencias por los cuatro estados. Por esta razón, el método es llamado algunas veces un método *decodificador contador*. Este método y el de contador de anillo se explicaron en el Capítulo 7 y conjuntamente en la Figura 7-22.

Control del PLA

El arreglo lógico programable fue introducido en la Sección 5-8. Se había mostrado en dicha sección que el PLA es un componente LSI que puede configurar cualquier circuito combinacional complejo. El control del PLA es esencialmente similar al registro de secuencia y al método del decodificador excepto que todos los circuitos combinacionales se configuran con un PLA, incluyendo el decodificador y la lógica de decisión. Es posible reducir el número de CI y el número de alambres de interconexión, usando un PLA para el circuito combinacional.

La Figura 10-4 muestra la configuración de un controlador PLA. Un registro de secuencia externo establece el estado presente del circuito de control. Las salidas PLA determinan cuáles microoperaciones deben iniciarse dependiendo de las condiciones de entrada externas y del estado presente del registro secuencial. Al mismo tiempo, otras salidas del PLA determinan el estado siguiente del registro de secuencia.

El registro de secuencia es externo al PLA si la unidad configura solamente circuitos combinacionales. Sin embargo hay algunos PLA disponibles que incluyen no solamente compuertas sino flip-flops dentro de la unidad. Este tipo de PLA puede configurar un circuito secuencial especificando las uniones que deben conectarse a los flip-flops de la misma manera que se especificaron las uniones de las compuertas.

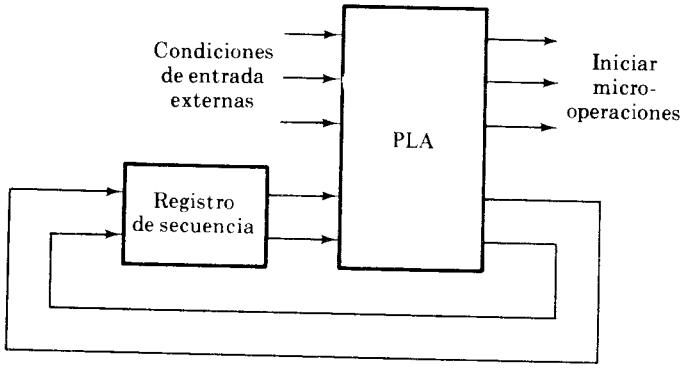


Figura 10-4 Lógica de control PLA

Control del microprograma

El propósito de la unidad de control es iniciar una serie de pasos secuenciales de microoperaciones. Durante cualquier tiempo dado se deben iniciar ciertas operaciones mientras que otras permanecen latentes. Así, las variables de control en un tiempo dado pueden ser representadas por una cadena de 1 ó 0 llamada *palabra de control*. Como tales, dichas palabras de control pueden ser programadas para iniciar las diferentes componentes en el sistema de una manera organizada. Una unidad de control cuyas variables de control se almacenan en una memoria, se llaman *unidad de control microprogramada*. Cada palabra de control de memoria se llama *microinstrucción* y una secuencia de microinstrucciones se llama *microprograma*. Como poco se necesitan las alteraciones del microprograma, la memoria de control puede ser una ROM. El uso del microprograma comprende la ubicación de todas las variables de control en palabras de la ROM para usarlas por medio de las unidades de control a través de operaciones sucesivas de lecturas. El contenido de la palabra en la ROM en una dirección dada especifica las microoperaciones del sistema.

Un desarrollo más avanzado, conocido como microprogramación *dinámica* permite cargar inicialmente un microprograma a partir de una consola de computador o de una memoria auxiliar tal como un disco magnético. Las unidades de control que usan microprogramación dinámica emplean una *memoria de control en la cual se puede escribir* (WCM = Writable control memory). Este tipo de memoria puede ser usada para escribir (o cambiar el microprograma) pero se usa mayormente para lectura. Una ROM, un PLA o un WCM cuando se usan en una unidad de control se los trata como *memoria de control*.

La Figura 10-5 ilustra la configuración general de la unidad de control de microprograma. La memoria de control se asume como una ROM dentro de la cual se almacena permanentemente toda la información de control. El registro de control de las direcciones de memoria especifica la palabra de control leída de la memoria de control. Se debe tener en cuenta que una ROM opera como un circuito combinacional con el valor de la dirección co-

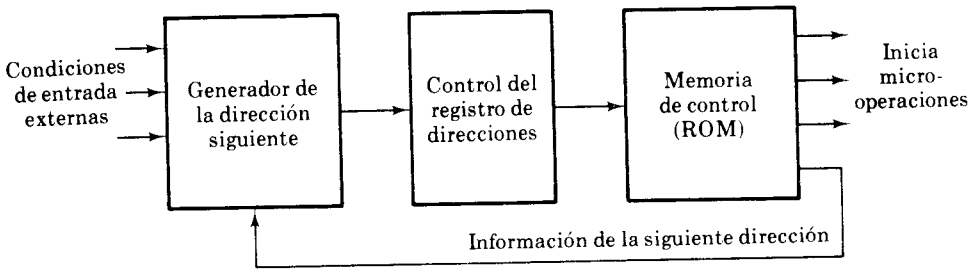


Figura 10-5 Lógica de control del microprograma

mo entrada y la palabra correspondiente como salida. El contenido de la palabra especificada permanece en los alambres de salida por el tiempo que el valor de la dirección permanece en el registro de dirección. No se necesita señal de lectura como en una memoria de acceso aleatorio. Una palabra que sale de la ROM debe trasferirse al registro separador, si el registro de direcciones cambia mientras que la palabra de ROM esté aún en uso. Si pueden ocurrir simultáneamente un cambio en dirección y una palabra de ROM no es necesario un registro separador.

La palabra leída de una memoria de control representa una microinstrucción. La microinstrucción especifica una o más microoperaciones para los componentes del sistema. Una vez que se ejecuten las operaciones, la unidad de control debe determinar la siguiente dirección. La ubicación de la siguiente microinstrucción podría ser la siguiente en secuencia o podría ser ubicada en otro lugar en la memoria de control. Por esta razón es necesario usar algunos bits de la microinstrucción para controlar la generación de la dirección para la siguiente microinstrucción. La siguiente dirección puede ser también una función de las condiciones de entrada externas. Mientras se ejecutan las microoperaciones, la siguiente dirección es computada en el circuito generador de la siguiente dirección y luego trasferida (con el siguiente pulso de reloj) al registro de control de direcciones para leer la siguiente microinstrucción. La construcción detallada del generador de la siguiente dirección depende de la aplicación particular.

El resto de este capítulo versa sobre ejemplos específicos del diseño de lógica de control. El primer ejemplo en la Sección 10-3 demuestra el método de un flip-flop por estado y la Sección 10-4 presenta el mismo ejemplo con un control microprogramado. La Sección 10-6 usa un segundo ejemplo para demostrar el método del registro de secuencia y el decodificador y la Sección 10-7 configura el segundo ejemplo con un PLA. Las Secciones 10-5 y 10-8 consideran el método de control del microprograma en más detalle.

10-3 CONTROL DE COMPONENTES ALAMBRADOS—EJEMPLO 1

Este ejemplo demuestra el desarrollo de un algoritmo de diseño. Se comienza con la proposición del problema y se procede con los pasos del diseño para obtener la lógica de control del sistema. El diseño se lleva a cabo en cinco pasos consecutivos.

1. Se enuncia el problema.
2. Se asume una configuración inicial del equipo.
3. Se formula el algoritmo.
4. Se especifica la parte del procesador de datos.
5. Se diseña la lógica de control.

Una configuración inicial del equipo es necesaria para poder formular el algoritmo diseñado en términos del método de trasferencia entre registros. El algoritmo se formula por medio del flujograma que especifica la secuencia de microoperaciones del sistema. Una vez que se tenga la lista de microoperaciones se pueden escoger funciones digitales necesarias para su configuración. En esencia, esto suministra la parte procesadora de datos del sistema. El control se diseña entonces para darle secuencia a las microoperaciones requeridas en el procesador de datos.

La lógica de control deducida en esta sección es un control de componentes alambrados por el método de un flip-flop por estado. El sistema digital presentado aquí se usa de nuevo en la siguiente sección para demostrar un ejemplo del control microprogramado.

Enunciado del problema

En la Sección 8-5 se enuncia un algoritmo para la adición y sustracción de los números binarios de punto fijo cuando los números negativos están en la forma de signo-complemento de 2. El problema aquí es configurar con materiales la adición y sustracción de dos números binarios de punto fijo representados en forma de signo-magnitud. Se puede usar aritmética complementada siempre y cuando el resultado final esté en la forma de signo-magnitud.

La suma de dos números almacenados en los registros de longitud finita podría resultar en una suma que excede la capacidad de almacenaje del registro en un bit. El bit extra se dice que causa sobrecapacidad. El circuito debe venir con un flip-flop para almacenar el bit de desbordamiento por sobrecapacidad.

Configuración del equipo

Los dos números binarios con signo al ser sumados o restados contienen n bits. Las magnitudes de los números contienen $k = n - 1$ bits y se almacenan en los registros A y B. Los bits de signo se almacenan en los flip-flops A_s y B_s. La Figura 10-6 muestra los registros y el equipo asociado. El ALU realiza las operaciones aritméticas y el registro E de 1 bit sirve como flip-flop de sobrecapacidad. El arrastre de salida del ALU se trasfiere al E.

Se asume que los dos números y sus signos han sido trasferidos a sus registros respectivos y que el resultado de la operación está disponible en los registros A y A_s. Las dos señales de entrada en el control especifican las operaciones de suma (q_a) y resta (q_s). La variable de salida x indica

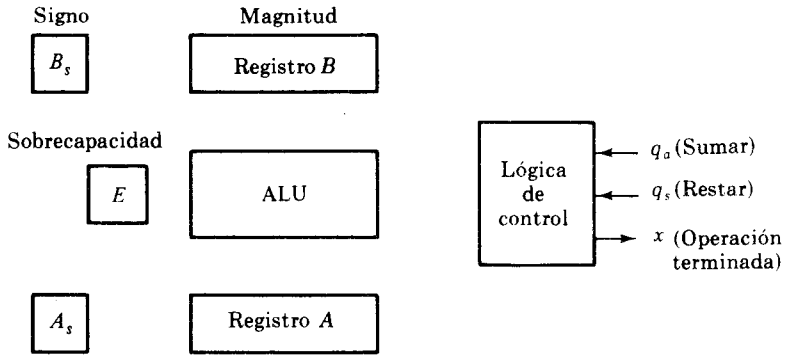


Figura 10-6 Configuración del registro para el sumador-sustractor

el final de la operación. La lógica de control se comunica con los circuitos que la rodean a través de las variables de entrada y salida. El control reconoce la señal de entrada q_a ó q_s y suministra la operación requerida. Una vez finalizada la operación, el control informa a los circuitos exteriores con la salida x que la suma o diferencia está en los registros A y A_s y el bit de sobrecapacidad está en E.

Deducción del algoritmo

La representación de números por medio de signo-magnitud es familiar debido a que se usa para los cálculos aritméticos a lápiz y papel. El procedimiento de sumar o restar dos números binarios con signo a lápiz y papel es muy simple y directo. Una revisión de este procedimiento podría ser útil para deducir el algoritmo diseñado.

Se designa la *magnitud* de dos números A y B. Cuando los números se suman o restan algebraicamente se encuentra que hay ocho condiciones diferentes para considerar, dependiendo del signo de los números y de la operación realizada. Las ocho condiciones pueden expresarse en forma compacta de la siguiente manera:

$$(\pm A) \pm (\pm B)$$

Si la operación aritmética especificada es la sustracción, se cambia el signo de B y se suma. Esto se hace evidente a partir de las relaciones:

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

Lo cual reduce el número de condiciones posibles a cuatro, a saber:

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

Cuando los signos de A y B son iguales, se agregan las dos magnitudes y el signo del resultado es el mismo que el signo común. Cuando los signos de

A y B no son iguales, se resta el número más pequeño del mayor y el signo del resultado es el signo del número mayor. Esto es evidente a partir de las siguientes relaciones:

	Si $A \geqslant B$	Si $A < B$
$(+A) + (+B) = +(A + B)$		
$(+A) + (-B) =$	$+(A - B) =$	$-(B - A)$
$(-A) + (+B) =$	$-(A - B) =$	$+(B - A)$
$(-A) + (-B) = -(A + B)$		

El flujograma de la Figura 10-7 muestra cómo se puede configurar una sustracción y una adición con signo magnitud con el equipo de la Figura 10-6. Se inicia una operación con la entrada q_s o la entrada q_a. La entrada q_s inicia una operación de sustracción de manera que se complementa el signo de B. La entrada q_a inicia una operación de suma y el signo de B se deja sin cambiar. El siguiente paso es comparar los dos signos. El bloque de decisión demarcado con $A_s : B_s$ simboliza esta decisión. Si los signos son iguales se sigue por el camino demarcado por el símbolo $=$; de otra manera se tomará el camino marcado por el símbolo $\neq$. El contenido de A se suma al contenido de B y la suma se trasfiere a A en el caso de símbolos iguales. El valor del arrastre final en este caso es una sobrecapacidad de manera que se hace el flip-flop E igual al arrastre de salida C_{out} . El circuito irá a su estado inicial y la salida x se convierte en 1. El signo del resultado en este caso es el mismo que el signo original A_s de manera que el bit de signo se deja sin cambiar.

Las dos magnitudes se restan si los signos no son iguales. La sustracción de las magnitudes se hace agregando A al complemento de 2 de B. No debe ocurrir sobrecapacidad si los dos números se sustraen de manera que E se lleva a 0. Un 1 en E indica que $A \geq B$ y el número en A es el resultado correcto. El signo del resultado es igual de nuevo al valor original de A_s. Un 0 en E indica que $A < B$. Para este caso es necesario formar el complemento de 2 del valor en A y el complemento del signo en A_s. El complemento de 2 de A puede hacerse con una microoperación $A \leftarrow \bar{A} + 1$. Sin embargo se requiere usar el ALU del Capítulo 9 y este ALU no tiene la operación de complemento de 2. Por esta razón el complemento de 2 se obtiene de las operaciones de complemento e incremento que están disponibles en el ALU.

Especificación del procesador de datos

El flujograma del algoritmo lista todas las microoperaciones para la parte del procesador de datos del sistema. Las operaciones entre A y B pueden hacerse con el ALU. Las operaciones con A_s, B_s y E deben ser iniciadas con variables de control separadas. La Figura 10-8(a) muestra el procesador de datos con las variables de control requeridas. Como se había men-

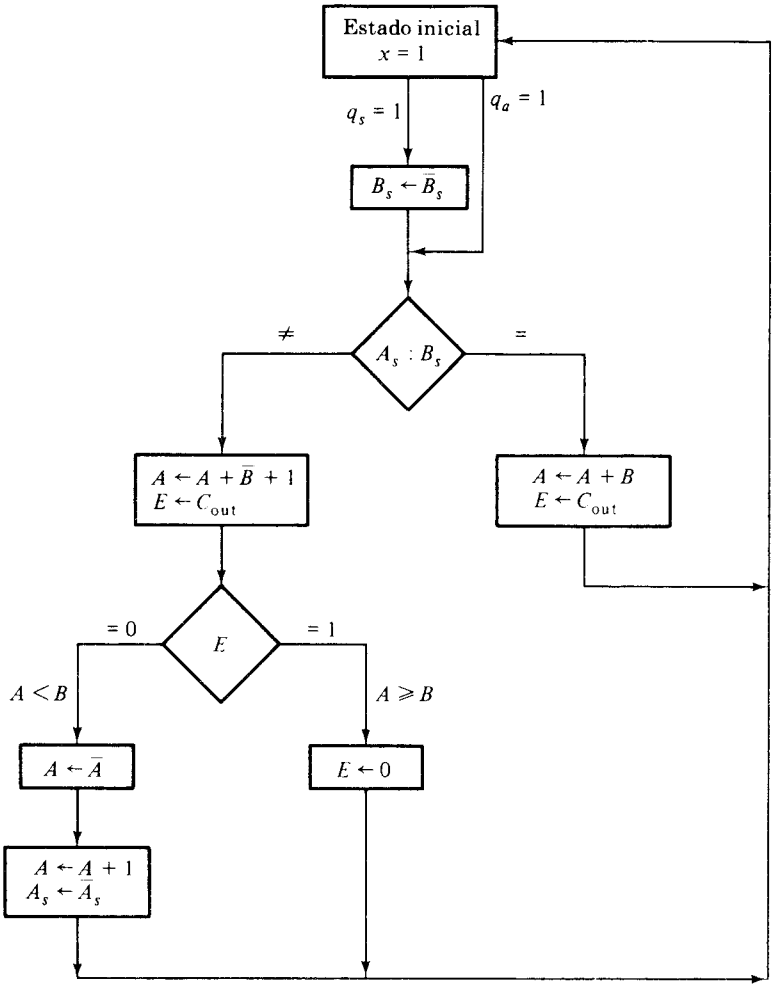
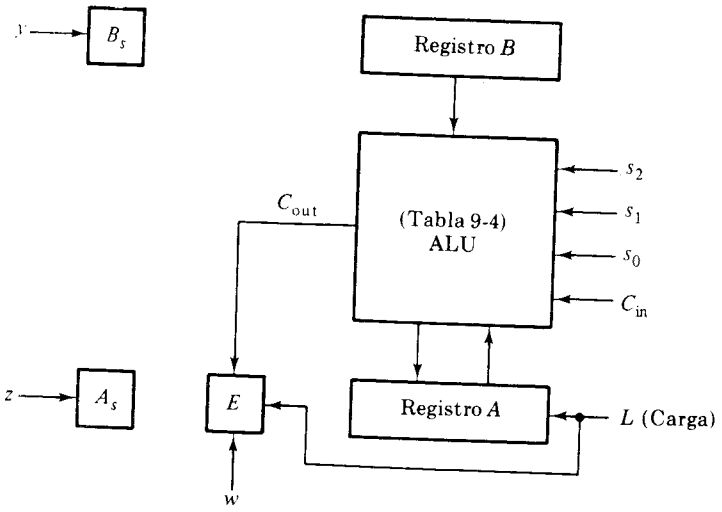


Figura 10-7 Flujograma para la adición y sustracción en signo-magnitud

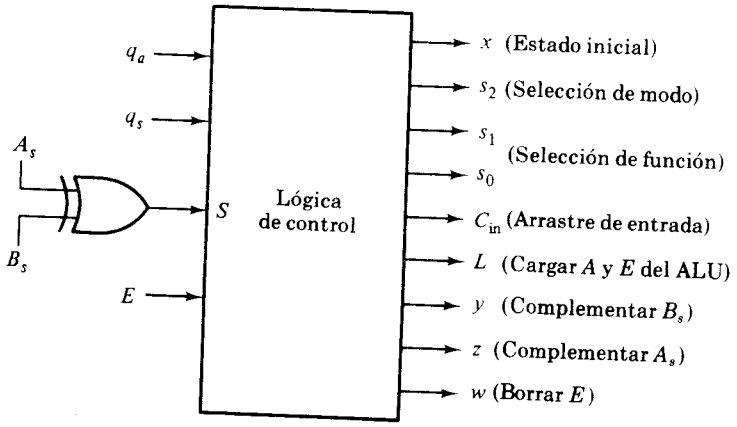
cionado antes el ALU viene del Capítulo 9 donde se especifica su tabla de función en la Tabla 9-4. Este ALU tiene cuatro variables de selección de la manera que se ilustra en el diagrama. La variable L carga la salida del ALU al registro A y también el arrastre de salida a E. Las variables y, z y w complementan B_s y A_s y borran E respectivamente.

El diagrama de bloque de la lógica de control se muestra en la Figura 10-8(b). El control recibe cinco entradas: dos de los componentes externos y tres del procesador de datos. Para simplificar el diseño se define una nueva variable S:

$$S = A_s \oplus B_s$$



(a) Registros procesadores de datos y ALU



(b) Diagrama de bloque del control

Figura 10-8 Diagrama de bloque del sistema

Esta variable da el resultado de la comparación entre dos bits de signo. La operación OR-exclusiva es igual a 1 si los dos signos no son iguales y es igual a 0 si los signos son ambos positivos o negativos.

El control suministra una salida x para el circuito externo. Esta selecciona también las operaciones en el ALU por medio de las cuatro variables de selección s_2, s_1, s_0 y C_{in}. Las otras cuatro salidas van a los registros en el procesador de datos como se especifica en el diagrama. Aunque no se

muestra en el diagrama, las salidas de la lógica de control deben estar conectadas a las correspondientes entradas en el procesador de datos. Ahora que se ha especificado el procesador se puede diseñar la lógica de control del sistema.

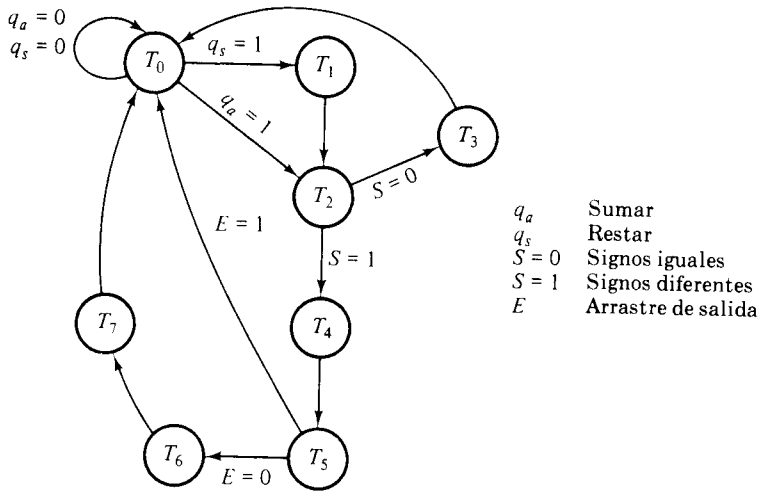
Diagrama de estado de control

El diseño de un control con materiales interconectados es un problema de lógica secuencial. Como tal, podría ser conveniente formular el diagrama de estado del control secuencial. Los recuadros de control en el flujograma pueden ser considerados como estados del circuito secuencial y los recuadros de decisión como condiciones del siguiente estado. Las microoperaciones que deben ser ejecutadas en un estado dado se especifican dentro del recuadro de función. Las condiciones para la transición del siguiente estado se especifican dentro del recuadro de decisión o en las líneas que se conectan entre dos recuadros de función. Aunque se puede formular esta relación entre un flujograma y el diagrama de estado, la conversión entre una forma y otra no es única. En consecuencia, diferentes diseñadores producirán diferentes diagramas de estado para el mismo flujograma y cada cual puede ser una correcta representación del sistema.

Se comienza asignando un estado incial T_0 al controlador secuencial. Se determina luego la transición a otros estados T_1, T_2, T_3 y así sucesivamente. Para cada estado se determinan las microoperaciones que deben iniciar el circuito de control. Este procedimiento produce el diagrama de estado para el controlador conjuntamente con una lista de operaciones de trasferencia entre registros, las cuales deben ser iniciadas mientras que el circuito de control esté en todos y cada uno de los estados.

El diagrama de control y las correspondientes operaciones de trasferencia entre registros se deducen en la Figura 10-9. La información para este diseño se toma directamente del flujograma de la Figura 10-7 y las variables definidas en el diagrama de bloque de la Figura 10-8. El estado de control inicial es T_0. Mientras que el control esté en ese estado la variable x se debe hacer igual a 1. Esta variable es 0 en todos los demás estados. Durante el tiempo en que q_a y q_s sean 0, el control permanecerá en su estado inicial. Si q_s se convierte en 1 el control realizará una operación de sustracción al pasar al estado T_1. En este estado el bit de signo B_s se complementa. El control pasa al estado T_2 para sumar los dos números. Si q_a se convierte en 1 el control irá directamente al estado T_2.

El siguiente estado después de T_2 depende de los valores relativos de los bits de signo, los cuales se determinan a partir de la variable S. Si los símbolos fueran iguales, S será 0 y el control pasará al estado T_3. En este estado, las dos magnitudes se suman y se pone a uno el bit de sobrecapacidad. Una vez que se haga lo anterior el control pasará a su estado inicial. Si los signos son diferentes, S es 1 y el control pasará del estado T_2 al estado T_4. En este estado las dos magnitudes se sustraen obteniendo el complemento de 2 de B. El arrastre final se trasfiere a E durante la sustracción y el control pasará al estado T_5.



(a) Diagrama de estado

	Salidas de control								
	x	s_2	s_1	s_0	C_{in}	L	y	z	w
T_0: Estado inicial $x = 1$	1	0	0	0	0	0	0	0	0
$T_1: B_s \leftarrow \overline{B}_s$	0	0	0	0	0	0	1	0	0
T_2: nada	0	0	0	0	0	0	0	0	0
$T_3: A \leftarrow A + B,\ E \leftarrow C_{out}$	0	0	0	1	0	1	0	0	0
$T_4: A \leftarrow A + \overline{B} + 1,\ E \leftarrow C_{out}$	0	0	1	0	1	1	0	0	0
$T_5: E \leftarrow 0$	0	0	0	0	0	0	0	0	1
$T_6: A \leftarrow \overline{A}$	0	1	1	1	0	1	0	0	0
$T_7: A \leftarrow A + 1,\ A_s \leftarrow \overline{A}_s$	0	0	0	0	1	1	0	1	0

(b) Secuencia de trasferencias del registro

Figura 10-9 Diagrama del estado de control y secuencia de microoperaciones

Se debe tener en cuenta que el arrastre final del ALU se trasfiere a E con un pulso de reloj. Esto sucede con el mismo pulso de reloj que causa que el control vaya del estado T_4 al T_5. Aunque se muestre la operación:

$$E \leftarrow C_{\text{out}}$$

conjuntamente con la variable T_4 esta operación no se ejecuta hasta que ocurra el pulso de reloj. Una vez que este pulso ejecute la operación, el control se encontrará en el estado T_5. Por tanto el valor de E para el arrastre final, no debe constatarse sino hasta que el control alcance el estado T_5. El valor de E se constata para determinar las magnitudes relativas de A y B. Si $E = 1$, esto indicará que $A \geq B$. Para este caso E debe borrarse para así completar la operación. Si $E = 0$, esto indicará que $A < B$. El control irá a los estados T_6 y T_7 para complementar A y A_s. Nótese que E se borra mientras que el control esté en el estado T_5. Esto se hace con E igual a 1 ó 0 ya que tratar de borrar un flip-flop que esté en 0 lo dejará de todas maneras en 0. Se debe notar también que E se borra con el pulso de reloj que causa que el control se salga del estado T_5. Se debe tener en cuenta que despejar E y trasferir el control al estado T_0 ó T_6 se hace con un pulso de reloj común sin problema. El valor original de E en el tiempo T_5 determina el siguiente estado aunque este flip-flop se borre mientras que el pulso de reloj pase por una transición de flanco.

Debe ser manifiesto con este ejemplo que la interpretación de un flujograma podría resultar en un diagrama de estado diferente para la misma lógica de control. Esto es aceptable siempre y cuando las restricciones de los materiales se tomen en consideración y el sistema funcione de acuerdo a las especificaciones. Por ejemplo, en vez de comprobar E en el tiempo T_5 se hubiera podido escoger el comprobar C_{out} en el tiempo T_4. Si C_{out} es 1, el control pasará al estado T_5 para despejar E. Si éste es 0, el control puede ir directamente al estado T_6, sin considerar el estado T_5 en este caso.

Diseño del control a base de componentes alambrados

Las salidas de control son función de los estados de control y se listan en la Figura 10-9(b). Estas salidas se definen en el diagrama de bloque de la Figura 10-8(b). Los valores para las variables de selección del ALU se determinan a partir de la Tabla 9-4. La variable L (cargar A) debe ser igual a 1 cada vez que la salida del ALU se trasfiera al registro A. De otra manera L es 0 y las salidas del ALU no tendrían efecto sobre el registro. Para diseñar el control de este sistema se necesita diseñar el diagrama de estado de la Figura 10-9(a) y dotarlo de las salidas de control como se especifican en la Figura 10-9(b).

El control puede diseñarse usando un procedimiento clásico lógico secuencial. Este procedimiento requiere una tabla de estado con ocho estados, cuatro entradas y nueve salidas. El circuito secuencial que se va a deducir de cada estado no será fácil de obtener debido a la gran cantidad de variables. El circuito obtenido, usando este método, puede tener un número mínimo de compuertas, pero tendrá un patrón irregular y será muy difícil de analizar en el caso de que ocurra una falla. Estas dificultades son eliminadas si se diseña el control por el método de un flip-flop por estado.

Una organización del control que use un flip-flop por estado tiene la característica conveniente de que el circuito puede deducirse directamente del diagrama de estado por inspección. No se necesitan tablas de estado o excitación si se usan flip-flops D. Recuérdese que el siguiente estado de un flip-flop D es una función de la entrada D y es independiente del estado presente. Como el método requiere un flip-flop por cada estado, se escogen ocho flip-flops D y se marcan sus salidas T_0, T_1, T_2, ..., T_7. La condición para poner a uno un flip-flop dado se especifica en el diagrama de estado. Por ejemplo, el flip-flop T_2 se pone a uno con el siguiente pulso de reloj si $T_1 = 1$ ó si $T_0 = 1$ y $q_a = 1$. Esta condición puede definirse con la función de Boole:

$$DT_2 = q_a T_0 + T_1$$

donde DT_2 designa la entrada D del flip-flop T_2. De hecho, la condición para poner a 1 flip-flop se obtiene de la condición especificada por las líneas de dirección que van a un estado de flip-flop dado y que a su vez se aplican conjuntamente con el estado previo del flip-flop a una función AND. Si hay más de una línea de dirección que va a un estado, todas las condiciones deben aplicarse a una función OR. Usando este procedimiento para otros flip-flops, se obtienen las funciones de entrada dadas en la Tabla 10-1.

Inicialmente, el flip-flop T_0 se pone a uno y las demás se borran. En un tiempo dado solamente una entrada D estará en el estado 1 mientras que las demás se mantienen en 0. El siguiente pulso de reloj pone a uno el flip-flop cuya entrada D es 1 y borra los demás. Por ejemplo si al presente $T_0 = 1$, entonces si $q_a = 0$ y $q_s = 0$, la entrada D de T_0 será 1 y el siguiente pulso dejará el flip-flop T_0 en el estado 1. Si durante el intervalo entre los dos pulsos q_s se convierte en 1, la entrada D de T_0 cambiará a 0 pero la entrada D de T_1 será 1, de manera que el siguiente pulso pondrá a uno T_1 y a cero T_0. Las funciones de entrada del flip-flop son mutuamente excluyentes y solamente un flip-flop puede ponerse a uno en un tiempo dado y los demás se borran porque sus entradas D son ceros.

Se necesita especificar las salidas de control como una función de los estados de los flip-flops. Esto se hace con las entradas de Boole dadas en la Tabla 10-1. Estas funciones de Boole se obtienen por inspección de la Fi-

Tabla 10-1 Funciones de Boole para control

Funciones de entrada de flip-flops	Funciones de Boole para el control de salida
$DT_0 = q_a' q_s' T_0 + T_3 + ET_5 + T_7$	$x = T_0$
$DT_1 = q_s T_0$	$s_2 = T_6$
$DT_2 = q_a T_0 + T_1$	$s_1 = T_4 + T_6$
$DT_3 = S' T_2$	$s_0 = T_3 + T_6$
$DT_4 = ST_2$	$C_{in} = T_4 + T_7$
$DT_5 = T_4$	$L = T_3 + T_4 + T_6 + T_7$
$DT_6 = E' T_5$	$y = T_1$
$DT_7 = T_6$	$z = T_7$
	$w = T_5$

gura 10-9(b). Por ejemplo la salida L debe ser 1 durante los estados T_3, T_4, T_6 ó T_7. Estas variables son disponibles en las salidas de los flip-flops. Lo que se necesita aquí es una compuerta OR de 4 entradas para generar el control de salida L.

El circuito para la lógica de control no se dibuja pero se puede obtener fácilmente de las funciones de Boole en la Tabla 10-1. Este circuito puede ser construido con ocho flip-flops D, siete compuertas AND, seis compuertas OR y cuatro inversores. Nótese que cinco salidas de control se toman directamente de las salidas de los flip-flops.

10-4 CONTROL DEL MICROPROGRAMA

En un microprograma de control, las variables de control que inician microoperaciones se almacenan en la memoria. La memoria de control es comúnmente una ROM ya que la secuencia de control es permanente y no necesita alteración. Las variables de control almacenadas en la memoria son leídas una a una para iniciar la secuencia de microoperaciones del sistema.

Las palabras almacenadas en la memoria de control son microinstrucciones y cada una de ellas especifica una o más microoperaciones para los componentes en el sistema. Una vez que se ejecutan estas microoperaciones, la unidad de control debe determinar la siguiente dirección. Por tanto, unos pocos bits de la microinstrucción se usan para controlar la generación de la dirección para la siguiente microinstrucción. Así una microinstrucción contiene bits para iniciar microoperaciones y bits para determinar la siguiente dirección para la memoria de control en sí misma.

Además de la memoria de control, una unidad de control de microprograma debe incluir circuitos especiales para seleccionar la siguiente dirección como se especifica por la microinstrucción. Estos circuitos y la configuración de los bits de microinstrucción almacenados en la memoria varían de una unidad a otra. En vez de profundizar en todas las posibilidades encontradas en las diferentes situaciones se escoge aquí introducir el concepto de microprograma por medio de un ejemplo simple.

La lógica de control que se va a diseñar es para el sumador-sustractor de signo-magnitud desarrollado en la sección anterior. El control a base de componentes conectados, diseñado en la Sección 10-3 será remplazado por un control de microprograma que se va a diseñar a continuación. Téngase en cuenta sin embargo que el sistema digital considerado aquí es muy pequeño para un controlador de microprograma y en la práctica un control a base de componentes conectados debe ser más eficiente. La organización del control de microprograma es más eficiente en sistemas mayores y complicados.

Un estado en la memoria de control se representa por la dirección de una microinstrucción. Una dirección para la memoria de control especifica una palabra de control dentro de una microinstrucción. El control que se desea diseñar se especifica en la Figura 10-9. Como hay ocho estados en el control se escoge una memoria de control con ocho palabras que tienen las direcciones 0 hasta 7. La dirección de la memoria de control corresponde al número suscrito bajo las T en el diagrama de estado.

La inspección del diagrama de estado revela que la secuencia de direcciones en el control del microprograma debe tener las siguientes cualidades:

1. Provisión para la carga de una dirección externa como resultado de la ocurrencia de las señales externas q_a y q_s.
2. Provisión para la sucesión consecutiva de direcciones.
3. Provisión para escoger entre dos direcciones como una función de los valores presentes de las variables de condición S y E.

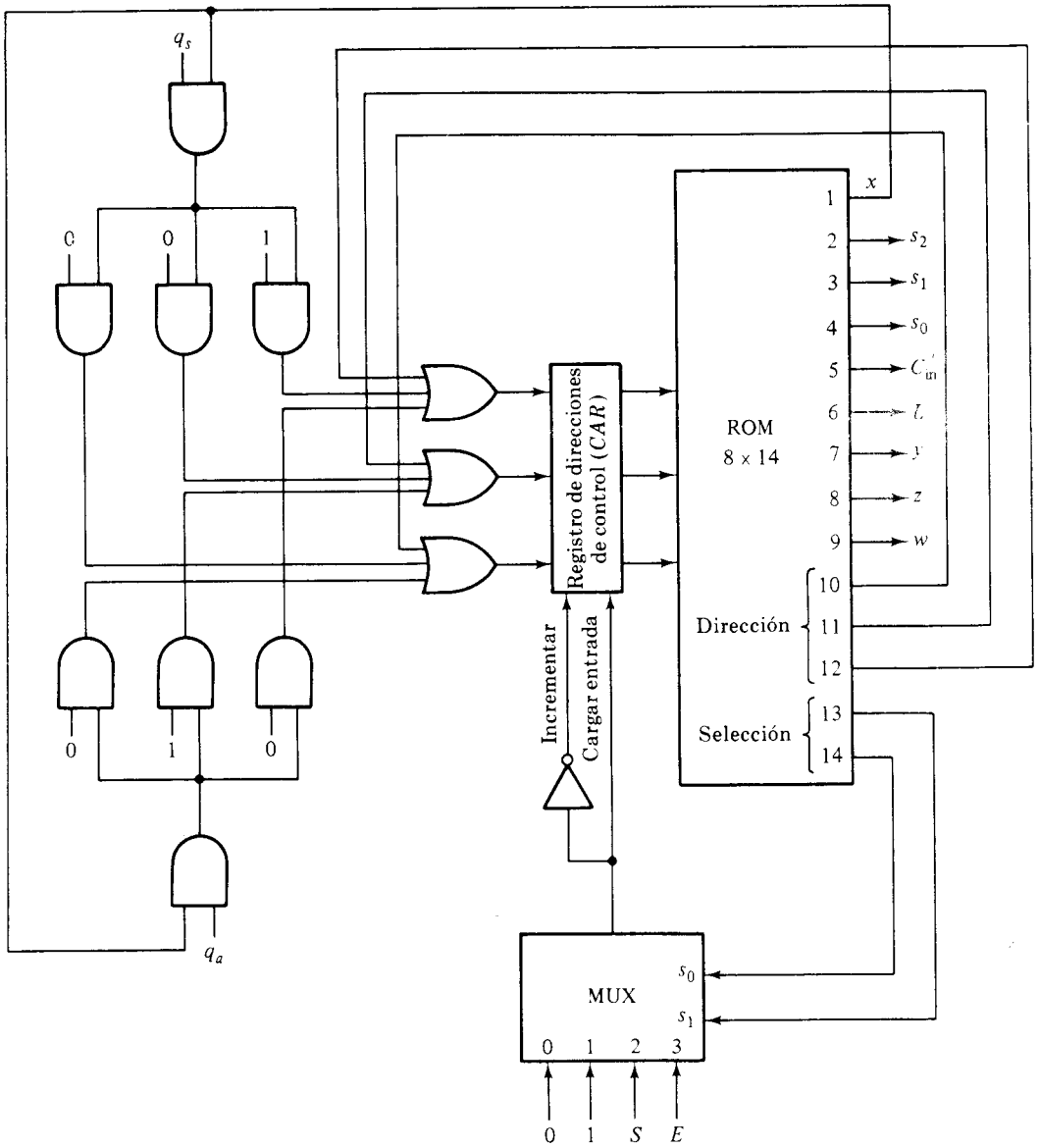
Cada microinstrucción debe contener un número de bits para especificar la manera en que se selecciona la nueva dirección.

Configuración de los materiales

La organización de la unidad de control del microprograma se muestra en la Figura 10-10. La memoria de control es una ROM de 8 palabras por 14 bits. Los primeros nueve bits de una palabra de microinstrucción contienen las variables de control que inician las microoperaciones. Los últimos cinco bits suministran información para seleccionar la siguiente dirección. El registro de direcciones de control (CAR = Control Address Register) almacena la dirección de la memoria de control. Este registro recibe un valor de entrada cuando se habilita su control de carga; de otra manera se incrementa en 1. Un CAR es esencialmente un contador con capacidad de carga en paralelo.

Los bits 10, 11 y 12 de una microinstrucción contienen una dirección para el CAR. Los bits 13 y 14 seleccionan una entrada para un multiplexor. El bit 1 suministra la condición de estado inicial denotada por la variable x y también habilita una dirección externa cuando q_s o q_a es igual a 1. Se estipula que cuando $x = 1$ el campo de dirección de la microinstrucción debe ser igual a 000. Entonces si $q_s = 1$, la dirección 001 está presente en las entradas del CAR, pero si $q_a = 1$, la dirección 010 se aplica al CAR. Si ambas q_s y q_a son cero, la dirección cero de los bits 10, 11 y 12 es aplicada a las entradas del CAR. De esta manera la memoria de control se mantiene en la dirección cero hasta que una variable externa se habilite.

El multiplexor (MUX) tiene cuatro entradas que se seleccionan con los bits 13 y 14 de la microinstrucción. Las funciones de los bits seleccionadas por el multiplexor se tabulan en la Figura 10-10. Si los bits 13 y 14 son 00, se selecciona una entrada de multiplexor que es igual a 0. La salida del multiplexor es 0 y la entrada de incremento al CAR se habilita. Esta configuración incrementa el CAR para escoger la siguiente dirección en secuencia. Una salida de 1 es seleccionada por el multiplexor cuando los bits 13 y 14 son iguales a 01. La salida del multiplexor es 1 y la entrada externa se carga al CAR. La variable de condición S es seleccionada cuando los bits 13 y 14 son iguales a 10. Si $S = 1$, la salida del multiplexor es 1 y los bits de dirección de la microinstrucción son cargados al CAR (si se tiene $x = 0$). Si $S = 0$, la salida del multiplexor es 0 y se incrementa el CAR. Con los bits 13 y 14 iguales a 11 se selecciona la variable de condición E y el campo de dirección se carga al CAR si $E = 1$; pero el CAR se incrementa si $E = 0$. Así, el



Bits de ROM 13	14	Función de selección del MUX
0	0	Incrementar el CAR
0	1	Cargar la entrada al CAR
1	0	Cargar las entradas al CAR si $S = 1$, incrementar el CAR si $S = 0$
1	1	Cargar las entradas al CAR si $E = 1$, incrementar el CAR si $E = 0$

Figura 10-10 Diagrama de bloque de control del microprograma

multiplexor permite al control escoger entre dos direcciones dependiendo del valor del bit de condición seleccionado.

El microprograma

Una vez que se establece la configuración de la unidad de control del microprograma la tarea del diseñador es generar el microcódigo para la memoria de control. Esta generación de código se llama microprogramación y es un proceso que determina la configuración de bits para cada una de las palabras en la memoria de control. Para apreciar este proceso, se deducirá el microprograma para el ejemplo del sumador-sustractor. La memoria de control tiene ocho palabras y cada palabra contiene 14 bits. Para microprogramar la memoria de control se debe determinar los valores de los bits de las ocho palabras.

El método de trasferencia entre registros puede ser adoptado para desarrollar un microprograma. La secuencia de microoperación puede ser especificada con declaraciones de trasferencia entre registros. No hay necesidad de listar las funciones de control con las variables de Boole ya que en este caso, las variables de control son las palabras de control almacenadas en la memoria de control. En vez de una función de control, se especifica una dirección con cada proposición de trasferencia entre registros. La dirección asociada con cada proposición simbólica corresponde a la dirección donde la microinstrucción es almacenada en la memoria. La secuencia de una dirección a la siguiente puede ser indicada por medio de proposiciones de control condicionales. Este tipo de proposiciones puede especificar una dirección a la cual va el control dependiendo de las condiciones establecidas. Así, en vez de pensar en términos de 1 ó 0 que deben ser agregados a cada microinstrucción, es más conveniente pensar en términos de símbolos en el método de trasferencia entre registros. Una vez que se ha establecido el microprograma simbólico, es posible trasladar las proposiciones de trasferencia entre registros o su forma binaria equivalente.

El microprograma se da en forma simbólica en la Tabla 10-2. Las ocho direcciones de la ROM se listan en la primera columna. La microinstrucción

Tabla 10-2 Microprograma simbólico para la memoria de control

Dirección de ROM	Microinstrucción	Comentarios
0	$x = 1$, si $(q_s = 1)$ entonces (va a 1), si $(q_a = 1)$ entonces (va a 2), si $(q_s \wedge q_a = 0)$ entonces (va a 0)	Cargar 0 o dirección externa
1	$B_s \leftarrow \overline{B}_s$	$q_s = 1$, comenzar sustracción
2	If $(S = 1)$ entonces (va a 4)	$q_a = 1$, comenzar suma
3	$A \leftarrow A + B$, $E \leftarrow C_{out}$, va a 0	Sumar magnitudes y regreso
4	$A \leftarrow A + \overline{B} + 1$, $E \leftarrow C_{out}$,	Sustraer magnitudes
5	Si $(E = 1)$ entonves (va a 0), $E \leftarrow 0$	Operación finalizada si $E = 1$
6	$A \leftarrow \overline{A}$	$E = 0$, complementar A
7	$A \leftarrow A + 1$, $A_s \leftarrow \overline{A}_s$, va a 0	Terminado, regresar a la dirección 0

que debe ser almacenada en cada dirección se da en forma simbólica en la segunda columna. Los comentarios se usan para clarificar las proposiciones de trasferencia entre registros. La dirección 0 es equivalente al estado inicial y produce una salida $x = 1$. La siguiente dirección depende de los valores de las variables externas q_s y q_a. Las tres proposiciones de control condicional en esta microinstrucción usan una proposición de *va a* después de la palabra *entonces*. Su significado se interpreta de manera que si la condición se satisface el control va a la dirección escrita después de la palabra *va a*. Así, si ambos q_s y q_a son cero el control permanece en la dirección cero para repetir la microinstrucción. Si q_s o q_a son 1, el control va a la dirección 1 ó 2 respectivamente.

Las proposiciones de control condicional en las otras microinstrucciones usan las variables de condición S y E. La proposición de *va a* sin una condición adjunta, especifica una alternativa incondicional a la dirección indicada. Por ejemplo, va a cero significa que el control va a la dirección cero después de ejecutar la microinstrucción presente. Si no hay una proposición de *va a* en la microinstrucción, esto implica que la siguiente microinstrucción se toma de la siguiente dirección de la secuencia. También, si la condición después de una proposición *si* no se satisface, el control va a la siguiente dirección en la secuencia.

Las microinstrucciones asociadas con las ocho direcciones se deducen directamente de las especificaciones de control de la Figura 10-9. Las microinstrucciones listadas son idénticas a aquellas listadas en la Figura 10-9(b). La proposición de control condicional especifica la secuencia de direcciones como se da en el diagrama de estado de la Figura 10-9(a). Nótese que cada número de dirección es igual al número suscrito bajo las T en el diagrama de estado. Debe ser obvio que las proposiciones de control condicional presentan una manera diferente de especificar el diagrama de estado. Esto muestra que el método de trasferencia entre registros puede ser usado para especificar un circuito secuencial.

El microprograma en la Tabla 10-2 se hubiera podido deducir directamente del flujograma de la Figura 10-7. Este flujograma se usa para especificar el algoritmo para el sistema que se está tratando de diseñar. Aunque el microprograma desarrollado aquí parece necesitar muchos pasos intermedios, se debe tener en cuenta que ello fue hecho así para propósitos de la explicación. Una vez que se entienda el concepto del microprograma no hay razón para no poder especificar el algoritmo directamente como un microprograma simbólico, sin la necesidad del diagrama de estado. Una vez que se ha establecido la configuración del equipo para el procesador de datos y el control del microprograma se puede desarrollar el algoritmo por medio de un microprograma.

La designación simbólica es un método conveniente para desarrollar el microprograma de una manera que la gente pueda leer y entender. Pero ésta no es la manera como el microprograma es almacenado en la memoria de control. El microprograma simbólico debe traducirse a binario porque ésta es la forma como va a la memoria. La traducción se hace dividiendo los bits de cada microinstrucción en sus partes funcionales llamadas *campos*. Aquí se tienen tres partes funcionales, los bits 1 hasta 9 especifican la palabra de control para iniciar las microoperaciones. Los bits 10 hasta 12 especifi-

Tabla 10-3 Microprograma binario para la memoria de control

Dirección de ROM			Salidas de ROM												
			x	s_2	s_1	s_0	C_{in}	L	y	z	w	*Dirección*			*Selección*	
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1
1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1

can un campo de dirección y los bits 13 y 14 seleccionan una entrada del multiplexor. Por cada microinstrucción que se lista en forma simbólica se deben escoger los bits adecuados en los campos de microinstrucción correspondientes.

La forma binaria equivalente del microprograma se da en la Tabla 10-3. Las direcciones para la memoria de control ROM se listan en binario. El contenido de cada palabra de ROM se da también en binario. Esta tabla constituye la tabla de verdad necesaria para programar la ROM.

Los primeros ocho bits en cada palabra ROM dan la palabra de control que inicia las microoperaciones especificadas. Estos valores de bit se toman directamente de la Figura 10-9(b). Los últimos cinco bits en cada palabra ROM se deducen de las proposiciones de control condicional en el programa simbólico.

En la dirección 000, se tiene 01 para el campo de *selección*. Esto permite que una dirección externa se cargue al CAR si q_s o q_a es igual a 1. De otra manera la dirección 000 se trasfiere al CAR. En la dirección 001, el campo de selección de la microinstrucción es 01 y el campo de dirección es 010. A partir de la tabla en la Figura 10-10, se encuentra que el pulso del reloj que inicia la microinstrucción $B_s \leftarrow \overline{B}_s$ (ya que $y = 1$) también trasfiere el campo de dirección al CAR. La siguiente microinstrucción que sale de la ROM será aquella almacenada en la dirección 010. El campo seleccionado en la dirección 001 podría haberse escogido como 00. Esto hubiera causado un incremento en el CAR para luego ir a la dirección 010.

La inspección de los campos de selección en los bits 13 y 14 muestra que cuando esos dos bits sean iguales a 01, el campo de dirección está en la nueva dirección. Cuando estos dos bits sean 10, se selecciona la variable de condición S y cuando sean 11 la variable E. En estos últimos dos casos, la siguiente dirección es aquella especificada en el campo de dirección si el bit de condición seleccionado es igual a 1. Si el bit de condición seleccionado es igual a 0, la siguiente dirección es la siguiente en secuencia ya que el CAR se incrementa.

10-5 CONTROL DE LA UNIDAD PROCESADORA

La configuración con materiales de la unidad de control del microprograma usada en la sección anterior es adecuada para el ejemplo particular considerado. En una situación práctica, la organización de los materiales de una unidad de control del microprograma debe tener una configuración de propósito general para adaptarse a una gran cantidad de situaciones. Una unidad de control de microprograma debe tener una memoria de control suficiente como para almacenar microinstrucciones. Se debe hacer provisión para incluir todas las variables de control posibles en el sistema y no solamente para controlar un ALU. El multiplexor y los bits seleccionados deben incluir todos los demás bits de condición posibles que se quieran comprobar en el sistema. Se debe tener una provisión para aceptar una dirección externa para iniciar muchas operaciones en vez de dos operaciones solamente tales como suma y sustracción.

La principal ventaja del control del microprograma es el hecho que una vez que se ha establecido la configuración de los materiales no debe haber necesidad de cambios posteriores de las conexiones entre los componentes. Si se quiere establecer una secuencia de control diferente para el sistema, todo lo que se necesita es especificar un conjunto diferente de microinstrucciones para la memoria de control. La configuración con los materiales no debe cambiar para las diferentes operaciones; el único cambio debe ser el microprograma que reside en la memoria de control.

Para demostrar la propiedad general de la organización del microprograma se expandirá la configuración de los componentes para incluir el control de toda una unidad de proceso. Una unidad procesadora de propósito general se introdujo en la Sección 9-9. Al referirse a la Figura 9-16, se nota que la unidad procesadora tiene siete registros, un ALU, un registro de desplazamiento y un registro de condición. Se selecciona una microoperación con una palabra de control de 16 bits. Los bits para una palabra de control dada pueden ser formulados del código binario que se lista en la Tabla 9-8.

Una organización del microprograma para controlar la unidad procesadora se muestra en la Figura 10-11. Esta tiene una memoria de control de 64 palabras, con 26 bits por palabra. Para seleccionar 64 palabras se necesita una dirección de 6 bits. Para seleccionar 8 bits de condición se necesitan 3 líneas de selección para el multiplexor. Un bit de la microinstrucción selecciona entre una dirección externa y el campo de dirección de la microinstrucción. Sumando los 16 bits para seleccionar la microoperación en el procesador se requiere un total de 26 bits por cada microinstrucción.

La unidad procesadora se incluye en el diagrama para mostrar sus conexiones a la unidad de control del microprograma. Los primeros 16 bits seleccionan la siguiente dirección para el control del registro de direcciones. Los bits de condición del procesador se aplican a las entradas del multiplexor. Se usan los dos valores normal y de complemento excepto para el bit de sobrecapacidad *V*. La entrada 0 del MUX 2 se conecta a una constante binaria la cual es siempre 1. La entrada de carga al *CAR* se habilita, cuando esta entrada es seleccionada por medio de los bits 18, 19 y 20 en la microinstrucción. Esto causa una trasferencia de información desde la sa-

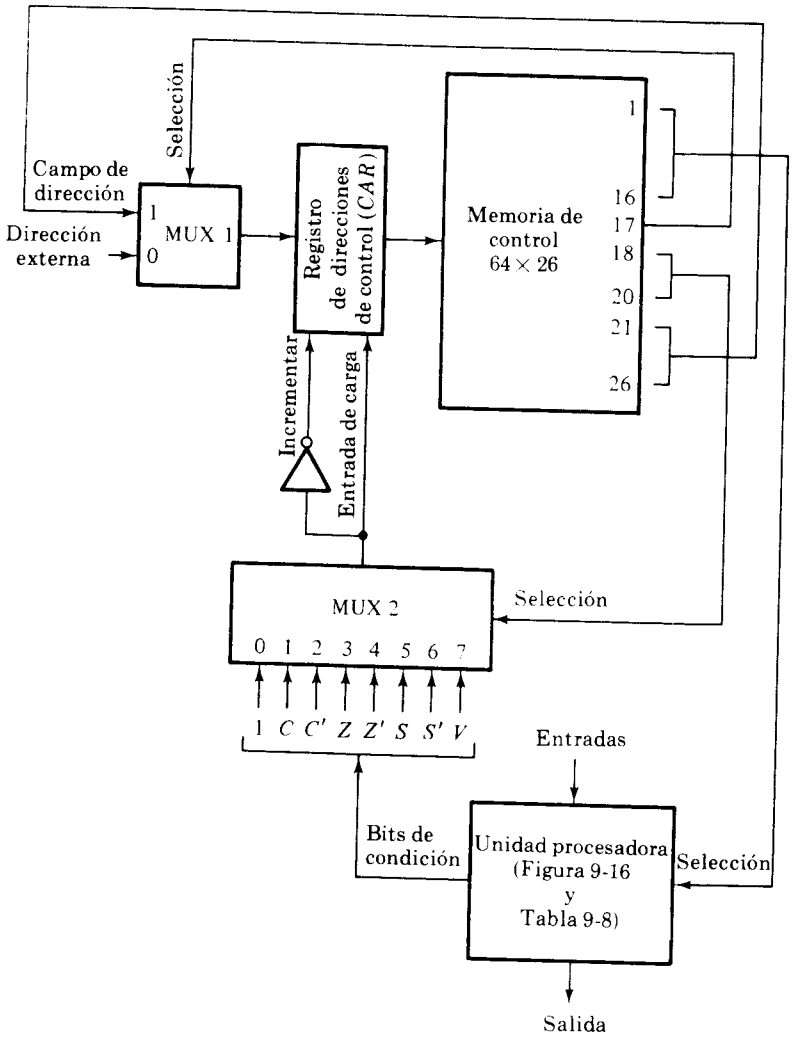


Figura 10-11 Control del microprograma para la unidad procesadora

lida del MUX 1 al *CAR*. La entrada al *CAR* es una función del bit 17 de la microinstrucción. Si el bit 17 es 1, el *CAR* recibe el campo de dirección de la microinstrucción. Si el bit 17 es cero se carga una dirección externa al *CAR*. La dirección externa es con el propósito de iniciar una nueva secuencia de microinstrucciones que pueden especificarse por los componentes externos. El bit de condición (o su complemento) seleccionado por los bits 18, 19, 20 de la microinstrucción puede ser igual a 1 ó 0. La dirección de entrada se carga al *CAR* si el bit seleccionado es 1, pero el *CAR* se incrementa si el bit seleccionado es 0.

Para construir microprogramas correctos es necesario especificar exactamente cómo el bit de condición es afectado por cada microoperación en el procesador. Los bits *S* (signo) y *Z* (cero) están afectados por todas las

operaciones. Los bits C (arrastre) y V (sobrecapacidad) *no cambian* después de las siguientes operaciones del ALU:

1. Las cuatro operaciones OR, AND, OR-exclusiva y complemento.
2. Las operaciones de incremento y decremento.

Para las demás operaciones, el bit de arrastre del ALU va al bit C del registro de condición. El bit C se afecta también después de un desplazamiento circular con operación de arrastre.

Ejemplo de microprograma

Se puede demostrar por medio de un ejemplo, cómo se escribe un microprograma para configurar una microoperación. Una microoperación inicia una secuencia de microinstrucciones en la memoria de control. Esta secuencia constituye una *rutina* de microprograma para ejecutar la macrooperación especificada. Una macrooperación se inicia por una dirección externa que aporta la primera dirección en la memoria de control para la rutina de microinstrucción. La rutina se termina con una microinstrucción que carga una nueva dirección externa para comenzar a ejecutar la siguiente macrooperación.

La macrooperación que se desea configurar cuenta el número de unos almacenados actualmente en el registro procesador $R1$ y alista el registro procesador $R2$ con ese número. Por ejemplo, si $R1 = 00110101$, la rutina del microprograma cuenta los cuatro unos almacenados en el registro y coloca en el registro $R2$ el número binario 100.

Aunque el microprograma puede deducirse directamente del enunciado del problema, sería conveniente construir un flujograma que muestre la secuencia de microoperaciones y vías de decisión. El flujograma para el microprograma se muestra en la Figura 10-12. Se asume que la rutina del microprograma comienza en la dirección 8. El registro $R2$ y el bit C (arrastre) se llevan primero a 0. Se examina entonces el contenido de $R1$. Si éste es 0 esto significa que no hay unos almacenados en él; de esta manera la rutina del microprograma finaliza con $R2$ igual a 0. Si el contenido de $R1$ no es 0, ello indica que hay algunos unos almacenados en él. El registro $R1$ conjuntamente con el arrastre se desplaza de manera circular cuantas veces sea necesario hasta que se trasfiera un 1 a C. Por cada 1 que se detecta en C, se incrementa el registro $R2$ y luego se comprueba si $R1$ es igual a 0. Este círculo se repite hasta que se hayan contado todos los unos de $R1$. Nótese que el valor de C es siempre 0 cuando se circula con lo contenido en $R1$.

La rutina del microprograma en forma simbólica se presenta en la Tabla 10-4. La rutina comienza en la dirección 8 borrando el registro $R2$. La microinstrucción de la dirección 9 borra el bit C y pone a uno el bit Z si el registro $R1$ contiene sólo ceros. Esto se hace trasfiriendo el contenido de $R1$ así mismo a través del ALU. La microinstrucción en la dirección 10 comprueba los valores del bit Z. Si éste es 1, es un indicio de que el registro $R1$ contiene sólo ceros y la rutina se termina aceptando una nueva dirección externa para comenzar a ejecutar otra macrooperación. Si Z no es igual a

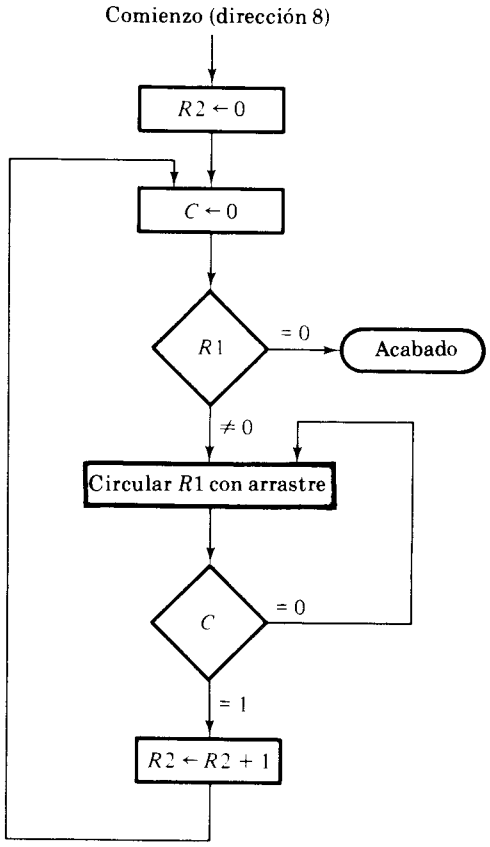


Figura 10-12 Flujograma para contar el número de unos en el registro $R1$

1 el control continúa con la dirección 11. El desplazamiento circular a la derecha con arrastre (crc) coloca el bit menos significativo de $R1$ en C. Luego se verifica el valor de C. Si éste es cero el control regresa a la dirección 11 para circular de nuevo hasta que C se convierta en 1. Cuando $C = 1$ el control pasa a la dirección 13 para incrementar $R2$ y luego regresará a la dirección 9 para comprobar el contenido de $R1$ para cada uno de los estados cero.

El microprograma binario se da en la Tabla 10-5. Los 16 bits para la palabra de control que seleccionan las microoperaciones del procesador se deducen de la Tabla 9-8. De hecho, la mayoría de las palabras de control listadas fueron explicadas en la Sección 9-9 conjuntamente con la Tabla 9-9. Los bits selectores del multiplexor seleccionan las entradas de los dos multiplexores. El bit 17 es 0 en la dirección 10 para seleccionar una dirección externa. En los demás casos es 1 para seleccionar el campo de dirección de la microinstrucción. Cuando los bits 18, 19 y 20 son 000 la siguiente dirección se determina directamente del campo de direcciones. Cuando estos bits son 011, éstos seleccionan el bit Z para el MUX 2. Si $Z = 1$ se

Tabla 10-4 Microprograma simbólico para contar el número de unos en $R1$

Dirección de ROM	Microinstrucción	Comentarios
8	$R2 \leftarrow 0$	Borrar el contador $R2$
9	$R1 \leftarrow R1, C \leftarrow 0$	Borrar C, poner a 1 los bits de condición
10	Si $(Z = 1)$ entonces (va a la dirección externa)	Acabar si $R1 = 0$
11	$R1 \leftarrow$ crc $R1$	Circular $R1$ a la derecha con arrastre
12	Si $(C = 0)$ entonces (va a 11)	Circular de nuevo si $C = 0$
13	$R2 \leftarrow R2 + 1$, va a 9	Arrastre = 1, incrementar $R2$

Tabla 10-5 Microprograma binario para contar el número de unos en $R1$

Dirección de ROM	Contenido del ROM												
	Selector de microoperación					Selector de MUX				Campo de dirección			
	A	*B*	*D*	*F*	*H*								
	1				16	17			20	21			26
001000	000	000	010	0000	011	1	0	0	0	0	0	1	0	0	1
001001	001	000	001	0000	000	1	0	0	0	0	0	1	0	1	0
001010	001	001	000	1000	000	0	0	1	1	0	0	0	0	0	0
001011	001	001	001	1000	101	1	0	0	0	0	0	1	1	0	0
001100	001	001	000	1000	000	1	0	1	0	0	0	1	0	1	1
001101	010	000	010	0001	000	1	0	0	0	0	0	1	0	0	1

trasfiere una dirección externa al CAR. Si $Z = 0$ se incrementa el CAR y la siguiente dirección será la siguiente en la secuencia. La microinstrucción en la dirección 12 selecciona el complemento del bit de arrastre o C'. Si $C = 0$, entonces $C' = 1$ y el campo de dirección (binario 1011) se trasfiere al CAR. Si $C = 1$, entonces $C' = 0$ y el CAR se incrementa para dar 13 para la siguiente dirección.

El lector familiarizado con la programación de máquina o lenguaje ensamblado para un computador, se dará cuenta que escribir programas es muy similar a escribir programas de lenguaje de máquina para un computador. Así, el concepto de microprograma es un procedimiento sistemático para diseñar la unidad de control de un sistema digital. Una vez que se haya establecido el formato de la microinstrucción, se hace el diseño escribiendo un microprograma, el cual es similar a escribir un programa para un computador. Por esta razón, el método del microprograma se refiere algunas veces como *firmware* para distinguirlo del método de los materiales (hardware o llamado también control de componentes conectados) y el concepto de *software* lo cual constituye un método de programación.

10-6 CONTROL A BASE DE COMPONENTES CONECTADOS—EJEMPLO 2

El ejemplo presentado en esta sección demuestra el desarrollo de un segundo algoritmo aritmético y de un método diferente para diseñar la lógica de control. Como en el ejemplo anterior, se desarrolla primero el algoritmo, diseñado conjuntamente con la configuración de los materiales, para la parte procesadora del sistema. Después de que se haga esto se formulan las especificaciones de la lógica de control del sistema.

La organización del control escogida para este ejemplo es el método del registro de secuencia y del decodificador. En la siguiente sección se diseña la lógica de control por medio de un PLA. Este ejemplo demuestra la relación directa que existe entre el registro de secuencia y el método del decodificador y su correspondiente configuración de control PLA.

Enunciado del problema

Se desea diseñar un circuito aritmético que multiplique dos números binarios de punto fijo representados en la forma signo-magnitud. El producto obtenido de la multiplicación de dos números binarios cuyas magnitudes consisten de k bits, podría tener hasta $2k$ bits. El signo de cada número ocupa un bit adicional.

La multiplicación de dos números binarios de punto fijo representados en la forma signo-magnitud se hace con lápiz y papel mediante sumas sucesivas y desplazamientos. Este proceso se ilustra de una mejor manera con un ejemplo numérico. Sea la multiplicación de los dos números binarios 10111 y 10011:

23		10111	multiplicando
	×		
19		10011	multiplicador
		10111	
		10111	
		00000	+
		00000	
		10111	
437		110110101	producto

Este proceso consiste en observar sucesivamente los bits del multiplicador con el bit menos significativo en primer lugar. Si el bit del multiplicador es 1 se copia el multiplicando; de lo contrario se copian sólo ceros. Los números escritos en líneas sucesivas se desplazan una posición a la izquierda del número previo. Finalmente, se agregan los números y su suma forma el producto.

El signo del producto se determina de los signos del multiplicando y multiplicador. Si son iguales el signo del producto será más. Si son diferentes el signo del producto será menos.

Cuando el proceso anterior se hace en una máquina digital es conveniente cambiar un poco el proceso. Primero, en vez de tener circuitos digitales que almacenen y sumen simultáneamente tantos números binarios como unos haya en el multiplicador, es conveniente tener circuitos para la suma de dos números binarios únicamente y sucesivamente acumular los productos parciales en el registro. Segundo, en vez de desplazar el multiplicando a la izquierda, el producto parcial se desplaza a la derecha lo cual da como resultado el colocar al producto parcial y al multiplicando en las posiciones relativas requeridas. Tercero, cuando el bit correspondiente del multiplicador es 0, no hay necesidad de agregar ceros al producto parcial ya que no se alterará su valor. El ejemplo numérico previo se repite aquí para clarificar el proceso de multiplicación propuesto:

multiplicando:	10111
multiplicador:	10011
1er. bit multiplicador = 1, copiar el multiplicando	10111
desplazamiento a la derecha para obtener el primer producto parcial	010111
2o. bit multiplicador = 1, copiar el multiplicando	10111
agregar el multiplicando al producto parcial previo	1000101
desplazamiento a la derecha para obtener el segundo producto parcial	1000101
3er. bit multiplicador = 0, desplazamiento a la derecha para obtener el tercer producto parcial	0100010i
4o. bit multiplicador = 0, desplazamiento a la derecha para obtener el cuarto producto parcial	001000101
5o. bit multiplicador = 1, copiar el multiplicando	10111
agregar el multiplicando al producto parcial previo	110110101
desplazamiento a la derecha para obtener el 5o. producto parcial = producto final	0110110101

Configuración del equipo

La configuración del registro para el multiplicador binario se muestra en la Figura 10-13. El multiplicando se almacena en el registro B, el multiplicador se almacena en el registro Q y el producto parcial se forma en el registro A. El signo del multiplicando está en B_s, el signo del multiplicador está en Q_s y el signo del producto se forma en A_s. El flip-flop E almacena el arrastre de salida después de la adición de B y A. Los dos números que se van a multiplicar consisten de n bits. Uno de estos bits contiene el signo y los restantes $k = n - 1$ contienen la magnitud del número. El contador P se prepa-

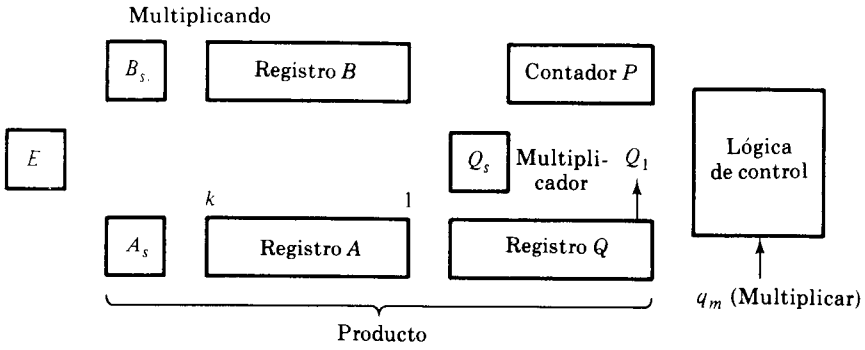


Figura 10-13 Registros para el multiplicador binario

ra con una cuenta o número binario igual al número de bits en la magnitud del multiplicador. Este contador se decrementa después de la formación de cada nuevo producto parcial. Cuando el contenido del contador llegue a cero, el producto estará formado en los registros A y Q y se suspenderá el proceso.

La lógica de control permanece en su estado inicial hasta que la variable q_m se convierta en 1. El control realizará entonces la multiplicación. La suma de A y B forma un producto parcial que se trasfiere a A. Si hay un arrastre producto de la suma, se trasfiere a E. El producto parcial en A y el multiplicador en Q son ambos desplazados a la derecha. Una vez que se haya realizado el desplazamiento a la derecha de A y Q, se trasfiere un bit del producto parcial a Q mientras que se desplazan los bits de Q una posición a la derecha. De esta manera el bit de la extrema derecha del registro Q, designado como Q_1, almacena siempre el bit del multiplicador, el cual debe ser inspeccionado en seguida.

Derivación del algoritmo

El flujograma del multiplicador binario se muestra en la Figura 10-14. Inicialmente el multiplicando está en B y el multiplicador en Q. Sus signos correspondientes están en B_s y Q_s. El proceso de multiplicación se inicia cuando $q_m = 1$. Se comparan los dos símbolos por medio de una compuerta OR-exclusiva. Si los dos signos son iguales la operación con la OR-exclusiva produce un 0 el cual se trasfiere a A_s, para dar un más para el producto. Si los signos son diferentes se trasfiere un 1 a A_s, para dar un signo negativo para el producto. Los registros A y E se borran y el contador de secuencia P se lleva a un número binario k, el cual es igual al número de bits en el multiplicador.

En seguida se cae en un bucle que continúa formando los productos parciales. Se comprueba el bit multiplicador en Q_1 y si se encuentra igual a 1 se agrega el multiplicando en B al producto parcial presente en A. Cualquier arrastre de la adición se trasfiere a E. El producto parcial en A se deja sin cambiar si $Q_1 = 0$. Se incrementa el contador P independientemente del

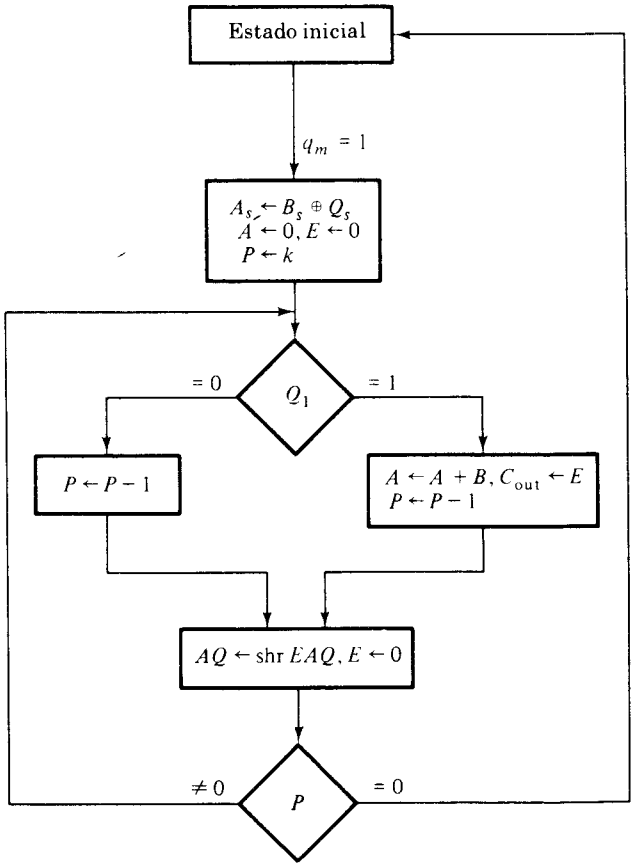


Figura 10-14 Flujograma para un multiplicador binario

valor de Q_1. Los registros A, Q y E de desplazan una vez a la derecha para obtener el nuevo producto parcial. Esta operación de desplazamiento se simboliza en el flujograma en forma compacta por medio de la proposición:

$$AQ \leftarrow \text{shr } EAQ, E \leftarrow 0$$

EAQ es un registro compuesto, hecho de los registros E, A y Q. Si se usan símbolos individuales para los registros, se puede describir la operación de desplazamiento por medio de las siguientes microoperaciones:

$$A \leftarrow \text{shr } A, Q \leftarrow \text{shr } Q, A_k \leftarrow E, Q_k \leftarrow A_1, E \leftarrow 0$$

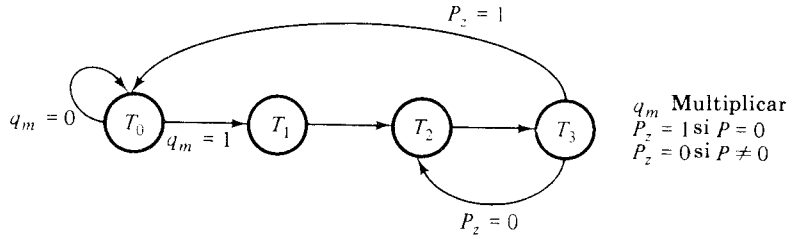
Ambos registros A y Q se desplazan a la derecha. La posición de la extrema izquierda de A designada como A_k recibe el arrastre de E. El bit de la extrema izquierda de Q ó Q_k recibe el bit que está en la posición de la extrema derecha de A en A_1 y se borra E. Se trata entonces de un desplazamiento largo del registro compuesto EAQ con un 0 colocado en la posición de la extrema izquierda localizada en E.

El valor del contador P se comprueba después de la formación de cada producto parcial. Si P no es cero, se repite el proceso y se forma un nuevo producto parcial. El proceso finaliza después del producto parcial k cuando $P = 0$. Nótese que el producto parcial formado en A se desplaza a Q bit a bit para eventualmente remplazar el multiplicador. El producto final estará disponible en A y Q con A conteniendo los bits más significativos y Q los menos significativos. El signo del producto estará en A_s.

Especificaciones del control

El algoritmo diseñado dado en el flujograma puede especificarse más precisamente por medio del diagrama de estado y una lista de operaciones de trasferencia entre registros. Se ha hecho mención previamente al hecho de que la conversión del flujograma al diagrama de estado no es única. El flujograma debe considerarse como una formulación preliminar del algoritmo. El diagrama de control de estado, conjuntamente con su lista de microoperaciones, es más preciso ya que toma en consideración las restricciones de los componentes del sistema.

La secuencia de control de las operaciones se define en la Figura 10-15. El control tiene cuatro estados y las operaciones de trasferencia entre registros de cada estado se listan a continuación del diagrama de estado. El control permanece en un estado inicial T_0 hasta que q_m se convierta en 1. Pasará luego al estado T_1 para inicializar los registros A, E y P y formar el signo del producto. El control pasa luego al estado T_2. En este estado se decrementa el registro P y el contenido de B se agrega a A si $Q_1 = 1$,



(a) Diagrama de estado

T_0: Estado inicial
T_1: $A_s \leftarrow B_s \oplus Q_s, A \leftarrow 0, E \leftarrow 0, P \leftarrow k$
$Q_1 T_2$: $A \leftarrow A + B, E \leftarrow C_{out}$
T_2: $P \leftarrow P - 1$
T_3: $AQ \leftarrow \text{shr}\, EAQ, E \leftarrow 0$

(b) Secuencia de trasferencias del registro

Figura 10-15 Diagrama de estado de control y secuencia de microoperaciones del multiplicador

de lo contrario A se deja sin cambiar. Las dos funciones de control en el tiempo T_2 son:

$$Q_1T_2: \quad A \leftarrow A + B, E \leftarrow C_{out}$$
$$T_2: \quad P \leftarrow P - 1$$

La segunda proposición se ejecuta siempre, cuando $T_2 = 1$. La primera proposición se ejecuta en el tiempo T_2 solamente si $Q_1 = 1$. Así, se puede incluir una variable de condición (Q_1 en este caso) con una variable de tiempo para formar una función de control. Nótese que es conveniente decrementar P en el estado T_2, de manera que su nuevo valor se pueda comprobar en el estado T_3.

El control pasará a T_3 después de T_2. En el estado T_3 el registro compuesto EAQ se desplaza a la derecha y el contenido de P se comprueba para buscar ceros. La variable binaria P_z es 1 si el registro P contiene sólo ceros, de lo contrario P_z es cero. Si $P_z = 1$ se termina la operación y el control pasa al estado inicial. Si $P_z = 0$ el control pasa al estado T_2 para formar el nuevo producto parcial. Nótese que P se refiere al contenido del registro mientras que P_z es la variable binaria.

Especificación del procesador de datos

La parte del procesador de datos del sistema puede derivarse de la lista de microoperaciones de la Figura 10-15(b). Un diagrama de bloque del procesador de datos se muestra en la Figura 10-16. Un sumador en paralelo se agrega entre los registros A y B para formar la suma, la cual es trasferida al registro A. El signo del producto se forma mediante una compuerta OR-exclusiva y la variable binaria P_z se genera con una compuerta NOR. Las salidas de la lógica de control inician las microoperaciones para el procesador de datos. La variable T_1 carga el signo del producto a A_s y el número k a P y borra los registros A y E. La variable T_2 decrementa el registro P y si $Q_1 = 1$ se genera la variable L, la cual carga la suma del sumador en paralelo a A y E. La variable T_3 desplaza A y Q a la derecha y borra E. La variable T_0 no tiene efecto sobre el procesador de datos ya que solamente indica que el sistema está en su estado inicial.

Las entradas a la lógica de control son las señales externas q_m y las dos condiciones de status P_z y Q_1. Las salidas son T_1, T_2, L y T_3. Aunque no se muestran en el diagrama, estas salidas deben estar conectadas a las correspondientes entradas en el procesador de datos. La compuerta AND que genera la variable L se muestra separadamente, aunque ésta sea parte de la lógica de control.

Diseño del circuito de control

La lógica de control para el multiplicador binario se especifica en el diagrama de estado de la Figura 10-15. El diagrama de estado tiene cuatro estados y dos entradas. Se necesitan dos flip-flops y un decodificador de 2×4

para configurarlo por medio del registro de secuencia y el método del decodificador. Aunque éste es un ejemplo sencillo, el procedimiento enunciado a continuación se aplica a situaciones más complicadas de la misma forma.

Se comienza con la tabla de excitación del circuito secuencial dado en la Figura 10-17(a). La parte de la tabla de estado se obtiene directamente del diagrama de estado. Las condiciones de entrada de los flip-flops son para dos flip-flops *JK* demarcadas G_1 y G_2. Nótese que la tabla de excitación tiene entradas de no importa en la mayoría de las entradas. Nótese también que un estado presente se lista más de una vez si éste tiene dos o más condiciones de estado siguiente. Las variables T_0 hasta T_3 se listan conjuntamente con las variables binarias para identificación. Las excitaciones de entrada a los flip-flops se obtienen directamente a partir de la tabla de excitación del flip-flop *JK* como se muestra en la Tabla 6-8(b).

El circuito secuencial puede ser diseñado de la tabla de excitación por medio de un procedimiento clásico. Este ejemplo tiene un pequeño número de estados y entradas; en la mayoría de aplicaciones de lógica de control, el número de estados y entradas es mucho mayor. La aplicación del método clásico requiere un cantidad excesiva de trabajo para obtener las funciones de entradas simplificadas para los flip-flops. El diseño puede ser simplificado si se tiene en consideración el hecho de que las salidas del decodificador están disponibles para ser usados en el diseño. En vez de usar las salidas de los flip-flops como condiciones de estado presente, se podría usar las salidas del decodificador para entregar esta información. Si las salidas del decodificador se designan por medio de las variables T_0, T_1, T_2, T_3 estas variables pueden ser usadas para suministrar las condiciones de estado presente del circuito.

Se podría decidir el obtener estas funciones directamente de la tabla de excitación, en vez de usar los mapas para la simplificación de las funciones de entrada de los flip-flops. Aunque puede que no se logre un circuito minimizado, el posible desperdicio de algunas compuertas se recompensa con el tiempo ahorrado. Por ejemplo, de la tabla de excitación se nota que la entrada *J* de G_2 (designada como JG_2 en la tabla) debe recibir un 1 binario solamente cuando la salida presente del decodificador es T_1. La entrada *K* de G_2 debe recibir un 1 cuando la salida del decodificador es T_3 siempre y cuando $P_z = 1$. Estas observaciones pueden escribirse en forma algebraica como:

$$JG_2 = T_1$$
$$KG_2 = T_3 P_z$$

En todos los demás casos, ambas entradas *J* y *K* de G_2 recibirán un 0 y el estado del flip-flop no cambiará. Esto es aceptable porque todas las demás entradas debajo de JG_2 y KG_2 tienen ceros o *X* de no importa.

De manera similar, es posible derivar las funciones de entrada del flip-flop para G_1 por imposición a partir de la tabla de excitación. Las funciones de entrada así obtenidas son:

$$JG_1 = T_0 q_m + T_2$$
$$KG_1 = 1$$

La razón para que KG_1 sea siempre 1 es que todas las entradas en la tabla para esta variable de entrada son 1 ó X.

No se puede estar seguro de que las funciones hayan sido simplificadas de la mejor manera posible cuando se derivan las funciones de entrada por inspecciones, partiendo de la tabla de excitación. Por esta razón se debería analizar siempre el circuito para asegurar que las ecuaciones derivadas producen en efecto las dos transiciones de estado requeridas de la manera especificada en la tabla de estado.

El diagrama lógico de la lógica de control se dibuja en la Figura 10-17 (c). Este consiste de dos flip-flops G_1 y G_2 y un decodificador. Las salidas del decodificador se usan para obtener el siguiente estado del circuito de acuerdo a las funciones de Boole listadas en la Figura 10-17(b). Las salidas del controlador deben estar conectadas a la parte del procesador de datos del sistema de la manera mostrada en la Figura 10-16.

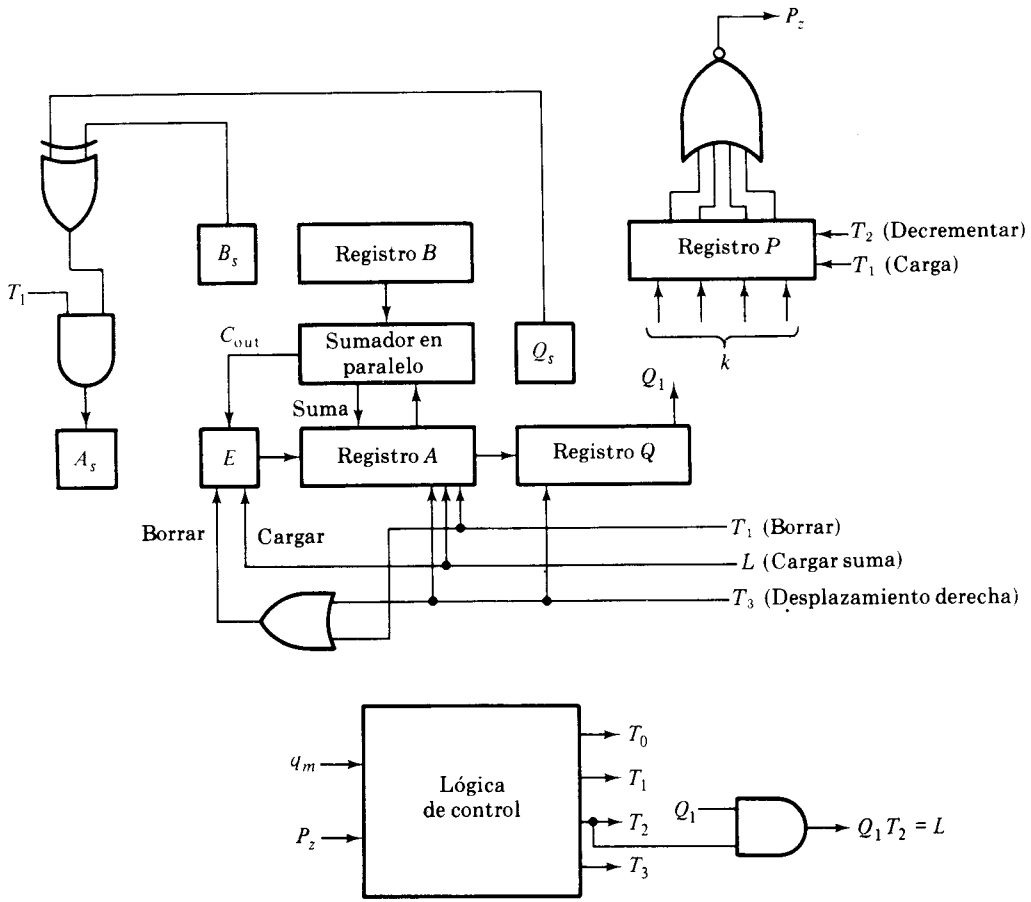


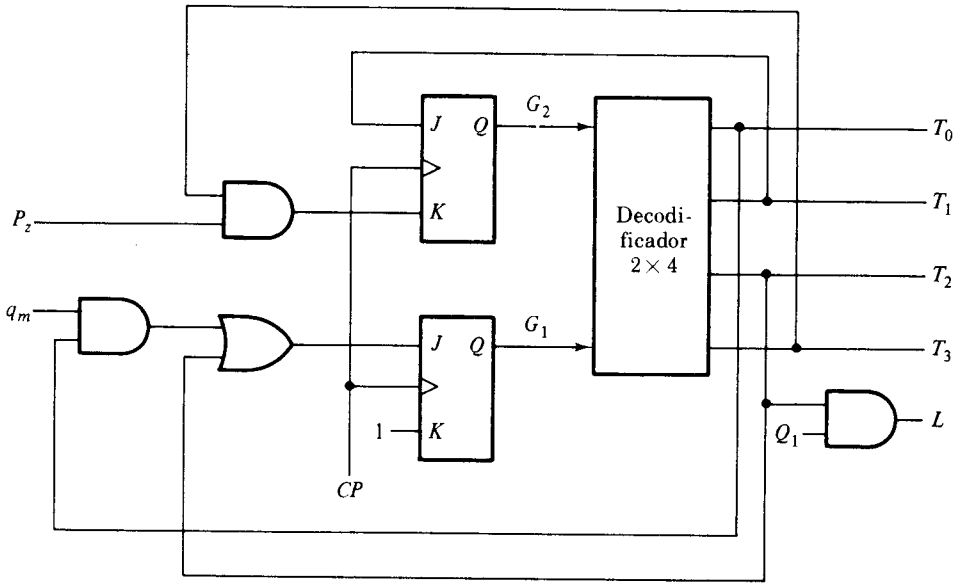
Figura 10-16 Procesador de datos para un multiplicador binario

	Estado presente		Entradas		Estado siguiente		Entradas del flip-flop			
	G_2	G_1	q_m	P_z	G_2	G_1	JG_2	KG_2	JG_1	KG_1
T_0	0	0	0	X	0	0	0	X	0	X
T_0	0	0	1	X	0	1	0	X	1	X
T_1	0	1	X	X	1	0	1	X	X	1
T_2	1	0	X	X	1	1	X	0	1	X
T_3	1	1	X	0	1	0	X	0	X	1
T_3	1	1	X	1	0	0	X	1	X	1

(a) Tabla de excitación

$$JG_2 = T_1 \qquad KG_2 = T_3P_z$$
$$JG_1 = T_0q_m + T_2 \qquad KG_1 = 1$$

(b) Funciones de entrada del flip-flop



(c) Diagrama lógico

Figura 10-17 Diseño del control para el multiplicador binario

10-7 CONTROL DEL PLA

Se ha visto de los dos ejemplos presentados en este capítulo que el diseño de un circuito de control es esencialmente un problema de diseño de lógica secuencial. En la Sección 7-2 se mostró que un circuito secuencial puede ser construido mediante un registro conectado a un circuito combinacional. En la Sección 5-8 se investigó el arreglo lógico programable y se demostró que puede usarse para configurar cualquier circuito combinacional. Es posible entonces diseñar un circuito de control con un registro conectado a un PLA simplemente remplazando el circuito combinacional con el PLA. El registro opera como un registro de secuencia que determina el estado del control. El PLA se programa para suministrar las salidas de control y del estado siguiente para dar secuencia al registro.

El diseño de una unidad de control con un PLA es muy similar al diseño que usa los métodos de registro de secuencia y decodificador. De hecho, el registro de secuencia en ambos métodos es el mismo. La diferencia de los métodos es la forma en que se configura la parte lógica combinacional del control. El PLA remplaza esencialmente el decodificador y otros circuitos lógicos de decisión necesarios para la configuración de los componentes.

La organización interna del PLA se presentó en la Sección 5-8. Se demostró también en dicha sección cómo obtener la tabla del programa del PLA. Se advierte al lector que sería conveniente repasar dicha sección para asegurar que se entiende el significado de la tabla de programa del PLA. Los caminos internos dentro del PLA son "programados" de acuerdo a las especificaciones dadas en la tabla de programa.

El diseño de un control PLA requiere que se obtenga la tabla de estado del circuito. El método del PLA debe usarse si la tabla de estado contiene muchas entradas de no importa, de lo contrario es más ventajoso usar una ROM en vez de un PLA. La tabla de estado da esencialmente toda la información requerida para obtener la tabla de programa del PLA (o la tabla de verdad de la ROM).

Tabla 10-6 Tabla de estado para el circuito de control

Estado presente		Entradas			Estado siguiente		Salidas				
G_2	G_1	q_m	P_z	Q_1	G_2	G_1	T_0	T_1	T_2	L	T_3
0	0	0	X	X	0	0	1	0	0	0	0
0	0	1	X	X	0	1	1	0	0	0	0
0	1	X	X	X	1	0	0	1	0	0	0
1	0	X	X	0	1	1	0	0	1	0	0
1	0	X	X	1	1	1	0	0	1	1	0
1	1	X	0	X	1	0	0	0	0	0	1
1	1	X	1	X	0	0	0	0	0	0	1

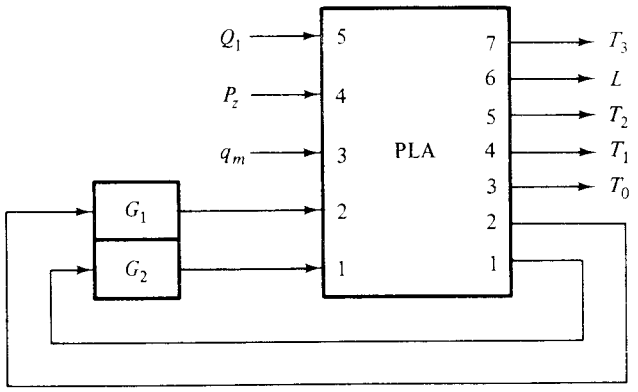
Para demostrar el procedimiento con un ejemplo, considérese el circuito de control para el multiplicador binario presentado en la sección previa. Las especificaciones de control para el multiplicador binario se dan en la Figura 10-15. A partir de esta información se obtiene la tabla de estado de la Tabla 10-6. El estado presente se determina a partir de los flip-flops G_1 y G_2. Las variables de entrada para el circuito de control son q_m, P_2 y Q_1. El siguiente estado de G_1 y G_2 puede ser una función de una de las entradas o puede ser independiente de cualquier entrada. Si una variable de entrada no influye en el estado siguiente, se marca con una X de no importa. Si el siguiente estado es una función de una entrada particular, el estado presente se repite en la tabla y a los estados siguientes se les asignan diferentes valores binarios. La tabla lista también todas las salidas de control como una función del estado presente y de las condiciones de entrada. Nótese que la entrada Q_1 no afecta al siguiente estado sino solamente determina el valor de la salida L cuando la salida T_2 es igual a 1.

El diagrama de bloque del control PLA se muestra en la Figura 10-18 (a). El PLA se conecta al registro de secuencia con dos flip-flops G_1 y G_2. Las entradas al PLA son los valores del estado presente del registro de secuencia y las tres entradas externas. Las salidas del PLA generan el estado siguiente para el registro de secuencia y las variables de salida de control. El estado presente del registro de secuencia, conjuntamente con las condiciones de entrada determinan en un tiempo dado los valores de salida y del siguiente estado para el registro de secuencia. El siguiente pulso de reloj inicia las microoperaciones especificadas por las salidas y trasfiere el estado siguiente al registro de secuencia. Esto suministra un nuevo estado de control y posibles valores de entrada diferentes. Así el PLA actúa como la parte de lógica combinacional de un circuito secuencial para entregar las salidas de control y los valores del siguiente estado del registro de secuencia.

Un PLA se especifica por medio del número de entradas, el número de términos de producto y el número de salidas. Para este caso se tienen cinco entradas y siete salidas. El número de términos producto es una función del circuito que se desea configurar.

La tabla de programa del PLA se puede obtener directamente a partir de la tabla de estado sin necesidad de procedimientos de simplificación. La tabla de programa del PLA en la Figura 10-18(b) especifica siete términos producto, uno para cada fila en la tabla de estado. Los terminales de entrada y salida se marcan con números y las variables aplicadas a esas terminales numeradas se indican en el diagrama de bloque. Los comentarios no constituyen parte de la tabla pero se incluyen para aclaraciones.

De acuerdo a las reglas establecidas en la Sección 5-8 una conexión para un camino de PLA se indica por medio de un guión (—) en la tabla. Las X en la tabla de estado designan las condiciones de no importa y no implican conexión para el PLA. Los 0 en las columnas de salida indican también no conexiones a las compuertas OR dentro del PLA. La conversión de la tabla de estado a una tabla de programa del PLA es muy simple: Las X de las columnas de entrada y los 0 en las columnas de salida se cambian a guiones y las demás entradas permanecen iguales. Nótese que las entradas al PLA son las mismas que las del presente estado y las entradas en la tabla de es-



(a) Diagrama de bloque

Término del producto	Entradas 1	2	3	4	5	Salidas 1	2	3	4	5	6	7	Comentarios
1	0	0	0	–	–	–	–	1	–	–	–	–	$T_0 = 1, \quad q_m = 0$
2	0	0	1	–	–	–	1	1	–	–	–	–	$T_0 = 1, \quad q_m = 1$
3	0	1	–	–	–	1	–	–	1	–	–	–	$T_1 = 1$
4	1	0	–	–	0	1	1	–	–	1	–	–	$T_2 = 1, \quad Q_1 = 0$
5	1	0	–	–	1	1	1	–	–	1	1	–	$T_2 = 1, \quad L = 1, \quad Q_1 = 1$
6	1	1	–	0	–	1	–	–	–	–	–	1	$T_3 = 1, \quad P_z = 0$
7	1	1	–	1	–	–	–	–	–	–	–	1	$T_3 = 1, \quad P_z = 1$

(b) Tabla del programa PLA

Figura 10-18 Control del PLA para un multiplicador binario

tado. Las salidas del PLA son las mismas que las del siguiente estado y las salidas de la tabla de estado.

El procedimiento para diseñar la lógica de control con un PLA debe ser evidente a partir de este ejemplo. De las especificaciones del sistema se obtiene primero la tabla de estado para el controlador. El número de estados determina el número de flip-flops para el registro de secuencia. El PLA se conecta al registro de secuencia y a las variables de entrada y salida. La tabla del programa del PLA se obtiene directamente de la tabla de estado.

La unidad PLA en un control PLA puede visualizarse como una memoria de control que almacena información de control para el sistema. Las salidas del registro de secuencia conjuntamente con las entradas externas, podrían considerarse como una dirección para tal memoria de control. Las salidas suministran una palabra de control para el procesador de datos y la información del estado siguiente especifica un valor parcial para la siguiente dirección en la memoria de control. Desde este punto de vista un control PLA puede clasificarse como una unidad de control de microprograma con el PLA remplazando la ROM para la memoria de control. Sin embar-

go, la organización de los dos métodos es diferente aunque hay una cierta cantidad de similitud entre el PLA y los métodos de control del microprograma.

Los ejemplos de control introducidos en este capítulo demuestran cuatro métodos de diseño de lógica de control. Estos no deberían considerarse los únicos métodos posibles. Un diseñador recursivo debe ser capaz de formular una configuración de control para adaptarse a una aplicación particular. Esta configuración puede consistir de una combinación de métodos o puede constituir una organización de control diferente de los que se han presentado aquí.

El diseño de la lógica de control para un computador digital sigue del mismo procedimiento que se enuncia en este capítulo. El papel del control del microprograma en la organización de un computador de propósito general se presenta en la siguiente sección. El Capítulo 11 presenta un diseño detallado de un computador digital y muestra cómo configurar su unidad de control por medio del método de conexión de componentes, el método del PLA y el método del microprograma.

10-8 SECUENCIADOR DEL MICROPROGRAMA

Una unidad de control de microprograma debe visualizarse como compuesta de dos partes: la memoria de control que almacena las microinstrucciones y los circuitos asociados que controlan la generación de la siguiente dirección. La parte de generación de dirección se llama algunas veces *secuenciador* de microprograma en vista de que da la secuencia de las microinstrucciones en la memoria de control. Un secuenciador de microprograma puede construirse con circuitos MSI para adaptarse a una aplicación particular. Sin embargo, de la misma manera que se encuentran disponibles en CI las unidades procesadoras para propósitos generales, se encuentran los CI para los secuenciadores para propósitos generales adecuados para la construcción de las unidades de control del microprograma. Para garantizar un amplio rango de aceptabilidad un secuenciador de CI debe suministrar una organización interna que pueda adaptarse a un amplio rango de aplicaciones.*

Un secuenciador de microprograma unido a la memoria de control inspecciona ciertos bits de la microinstrucción, de los cuales se determina la siguiente dirección para el control de la memoria. Un secuenciador típico presenta las siguientes características de secuencimiento de direcciones:

1. Incrementa la dirección presente para la memoria de control.
2. Se ramifica a una dirección como se especifica en el campo de dirección de la microinstrucción.
3. Se ramifica a una dirección dada, si el bit de condición especificado es igual a 1.

*Algunos secuenciadores comerciales son CI tipo 8 × 02 (Signetics) 9408 (Fairchild) y 2910 (Advanced Micro Devices).

4. Trasfiere el control a una nueva dirección de la manera especificada por una fuente externa.
5. Tiene la facilidad para hacer subrutinas con llamadas y retornos.

En la mayoría de los casos se leen las microinstrucciones de la memoria de control en sucesión. Este tipo de secuencia puede lograrse fácilmente incrementando el registro de dirección de la memoria de control. En algunos formatos de microinstrucción, cada microinstrucción contiene un campo de dirección aún para direcciones secuenciales. Esto elimina la necesidad de incrementar el registro de dirección de la memoria de control porque el campo de dirección disponible en cada microinstrucción especifica la dirección de la siguiente microinstrucción. En cada caso, se debe dejar la alternativa de ramificarse a una dirección que esté por fuera de la secuencia normal.

El control debe trasferirse de vez en cuando a una microinstrucción no secuencial, de manera que el secuenciador debe suministrar la capacidad de ramificarse a cualquiera de las dos direcciones dependiendo de si el bit de condición es 0 ó 1. La manera más sencilla de lograr lo anterior es ramificándose a la dirección especificada por el campo de dirección de la microinstrucción si el bit de condición especificado es igual a 1 o si no pasar a la siguiente dirección en secuencia si el bit de condición es igual a 0. Esta configuración requiere la capacidad de incrementar el registro de dirección.

El secuenciador trasfiere una nueva dirección para que la memoria de control comience a ejecutar la nueva microoperación. La dirección externa trasfiere el control a la primera microinstrucción en una rutina de microprograma que ejecuta la macrooperación especificada.

Las *subrutinas* son programas usados por otras rutinas para lograr una tarea dada. Las subrutinas pueden llamarse desde cualquier punto dentro del cuerpo principal del microprograma. Frecuentemente muchos microprogramas contienen secciones idénticas de código. Las microinstrucciones pueden conservarse usando subrutinas que usan secciones comunes de microcódigo. Los microprogramas que usan subrutinas deben tener una provisión para almacenar direcciones de retorno durante una llamada de subrutina y restaurar la dirección durante una subrutina de retorno. Esto puede lograrse colocando la dirección de retorno en un registro especial y luego ramificarse al comienzo de la subrutina. Este registro especial puede entonces convertirse en la fuente de dirección para alistar el registro de dirección para el regreso a la subrutina principal. La mejor manera de organizar un archivo de registro que almacene direcciones para llamadas de subrutinas y que regrese, es usar una pila (LIFO) último en entrar primero en salir. La organización de la pila y su uso en las llamadas de subrutina y regreso se explica en mayor detalle en la Sección 12-5.

El diagrama de bloque de un secuenciador de microprograma se muestra en la Figura 10-19. Este consiste de un multiplexor que selecciona una dirección de cuatro fuentes y la dirige al registro de dirección de control. La salida del *CAR* suministra la dirección para la memoria de control. El contenido del *CAR* se incrementa y aplica al multiplexor y al archivo de registro de la pila. El registro seleccionado de la pila se determina por me-

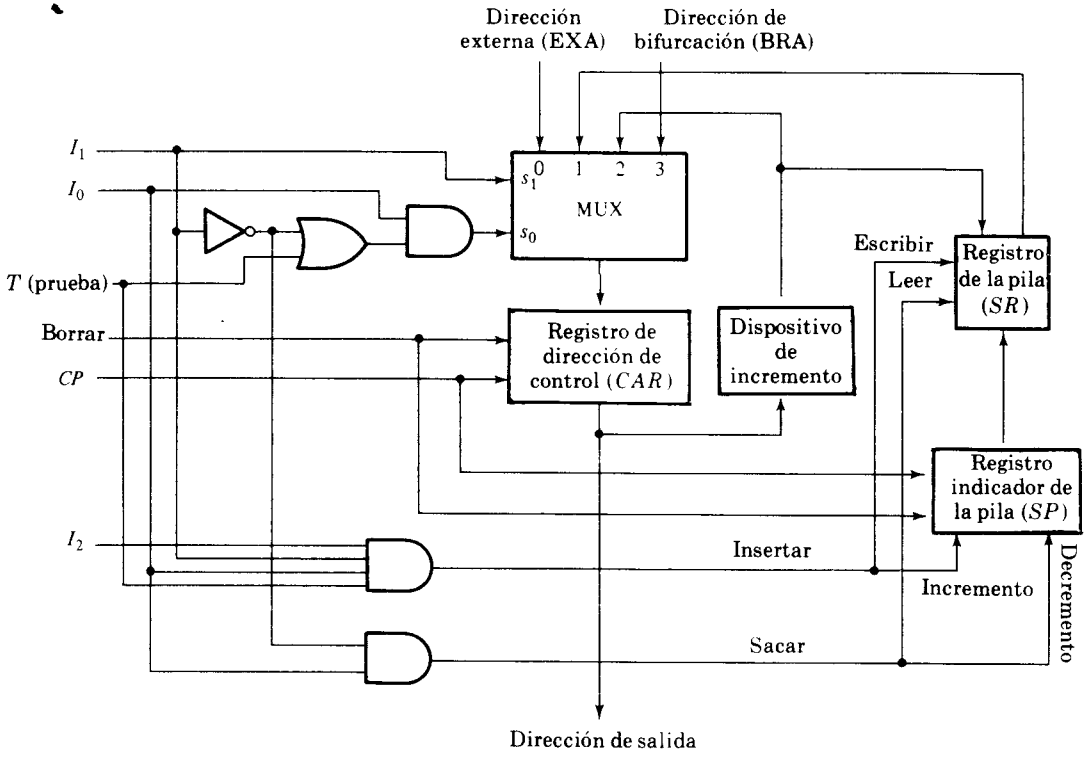


Tabla de función

I_2	I_1	I_0	T	s_1	s_0	Operación	Comentarios
X	0	0	X	0	0	$CAR \leftarrow EXA$	Trasferir la dirección externa
X	0	1	X	0	1	$CAR \leftarrow SR$	Trasferir de la pila de registro
X	1	0	X	1	0	$CAR \leftarrow CAR + 1$	Incrementar dirección
0	1	1	0	1	0	$CAR \leftarrow CAR + 1$	Incrementar dirección
0	1	1	1	1	1	$CAR \leftarrow BRA$	Trasferir dirección de bifurcación
1	1	1	0	1	0	$CAR \leftarrow CAR + 1$	Incrementar dirección
1	1	1	1	1	1	$CAR \leftarrow BRA$, $SR \leftarrow CAR + 1$	Bifurcar a la subrutina

Figura 10-19 Organización de un secuenciador de microprograma típico

dio del registro indicador de la pila (stack pointer). Las entradas I_0, I_1 e I_2 especifican la operación para el secuenciador y la entrada T es el punto de prueba para el bit de condición. El registro de dirección puede llevarse a cero para iniciar el sistema y los pulsos de reloj sincronizan la carga a los registros.

La tabla de función listada en el diagrama especifica la operación del secuenciador. Las entradas I_1 e I_0 determinan las variables de selección para el multiplexor. Una dirección externa (EXA) se trasfiere al CAR cuando $I_1 I_0 = 00$. La trasferencia del registro de la pila (SR) ocurre cuando

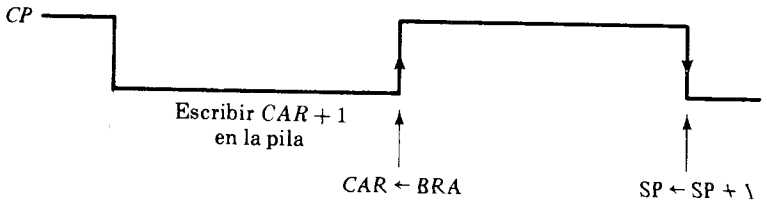
$I_1I_0 = 01$ y el *CAR* se incrementa cuando $I_1I_0 = 10$. Las entradas T e I_2 no tienen efecto durante estas tres operaciones y se marcan con entradas X de no importa. Cuando $I_1I_0 = 11$, el secuenciador ejecuta una operación de ramificación condicional dependiente del valor del bit de prueba en T. Si I_2 es también igual a 1, la operación es una llamada condicional a la subrutina. En cada caso, el *CAR* se incrementa si el bit de prueba T es 0. La dirección de ramificación (*BRA*) se trasfiere al *CAR* si $T = 1$. Así, con $I_1I_0 = 11$, el secuenciador se ramifica al *BRA* si el bit de condición en T es igual a 1 o incrementa el *CAR* si el bit de condición es cero. La dirección de ramificación normalmente viene del campo de la dirección de la microinstrucción.

La llamada de subrutina condicional ($I_2 = 1$) es similar a la ramificación condicional ($I_2 = 0$), excepto que la primera use la pila y la última no. La dirección almacenada en la pila durante una llamada de subrutina se toma del dispositivo de incremento. Este es la siguiente dirección en secuencia y es llamada la *dirección de regreso*. La dirección de regreso se trasfiere de nuevo al *CAR* con una operación de regreso a la subrutina ($I_1I_0 = 01$).

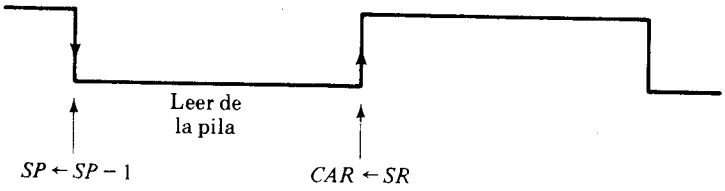
La operación del registro de la pila y el indicador de pila (stack pointer) será entendida de mejor manera después de leer la Sección 12-5. Un registro pila (o memoria) es similar a la unidad de memoria excepto que la dirección para la pila se determina a partir del valor en el registro del indicador de pila (stack pointer register). El acceso a la pila está en la secuencia de último en entrar primero en salir y se controla incrementando o decrementando el indicador de la pila. Inicialmente el indicador de pila se borra y se dice que *apunta* a la dirección 0 en la pila. La *escritura* o la trasferencia de información a la pila se llama *inserción* (push). Esto consiste de escribir la información de entrada en la pila en la dirección especificada por el indicador de pila para luego incrementar el registro del indicador de pila. De esta manera se trasfiere información a la pila y el indicador de pila indica el siguiente lugar libre en la pila. La *lectura* o la trasferencia de información hacia afuera de la pila se llama *sacar* (pop). Esta consiste de decrementar primero el registro del indicador de pila y luego leer el contenido del registro (o palabra) especificado por el nuevo valor del indicador de pila.

Una llamada de subrutina se ejecuta cuando $I_2I_1I_0 = 111$ y $T = 1$. Esta causa una operación de *sacar de la pila* (push-stack) y una ramificación a la dirección especificada por el *BRA*. Esto se configura almacenando primero el valor incrementado del *CAR* a la pila. Cuando el pulso de reloj *CP* pasa a través de una transición de flanco positivo, la dirección *BRA* se trasfiere al *CAR* y se inhibe la entrada de lectura a la pila. El registro del indicador de la pila se incrementa posteriormente cuando *CP* pase por su transición de flanco negativo. Esto se ilustra en la Figura 10-20(a).

El regreso de la subrutina se ejecuta cuando $I_1I_0 = 01$. Este causa una operación de *sacar de la pila* y una ramificación a la dirección almacenada en la cima de la pila. Esto se configura decrementando primero el registro indicador de la pila en la transición del flanco negativo del *CP*. El valor en la pila, dado por la dirección contenida al presente en el indicador de pila, se lee y trasfiere al *CAR* en la transición del flanco positivo del *CP*. Esto se ilustra en la Figura 10-20(b). Nótese que el *CAR* se dispara durante el flan-



(a) Subrutina de llamado (insertar a la pila) $I_2 I_1 I_0\, T = 1111$



(b) Regreso de la subrutina (sacar de la pila) $I_1 I_0 = 01$

Figura 10-20 Operaciones de la pila en el secuenciador de microprograma

co positivo y SP durante el flanco negativo de un pulso de reloj. El indicador de la pila (SP) se incrementa después de la trasferencia al CAR y se decrementa antes de trasferirse al CAR.

Organización del CPU microprogramado

Un computador digital consiste de una unidad procesadora central (CPU), una unidad de memoria y un dispositivo de entrada-salida. El CPU puede ser clasificado en dos secciones funcionales interactivas y diferentes. Una es la sección de proceso y la otra es la sección de control. Una unidad procesadora es un dispositivo útil para construir la sección del procesador del CPU. El secuenciador del microprograma es un elemento conveniente para construir un control del microprograma para el CPU. Se va a desarrollar ahora un computador CPU para mostrar la utilidad del secuenciador de microprograma definido en la Figura 10-19.

Un diagrama de bloque de un computador microprogramado se muestra en la Figura 10-21. Consiste de una unidad de memoria, dos unidades procesadoras un secuenciador de microprograma, una memoria de control y varias funciones digitales. Esta configuración puede ser comparada con el computador sencillo que fue diseñado en la Sección 8-9 y cuyo diagrama de bloque se da en la Figura 8-16.

La unidad de memoria almacena las instrucciones y datos suministrados por el usuario a través de un dispositivo de entrada. El procesador de datos manipula los datos y el procesador de direcciones manipula las direcciones recibidas de la memoria. Los dos procesadores pueden combinarse en una unidad, pero algunas veces es conveniente separarlos para poder suministrar un bus diferente para la dirección de la memoria. Una instruc-

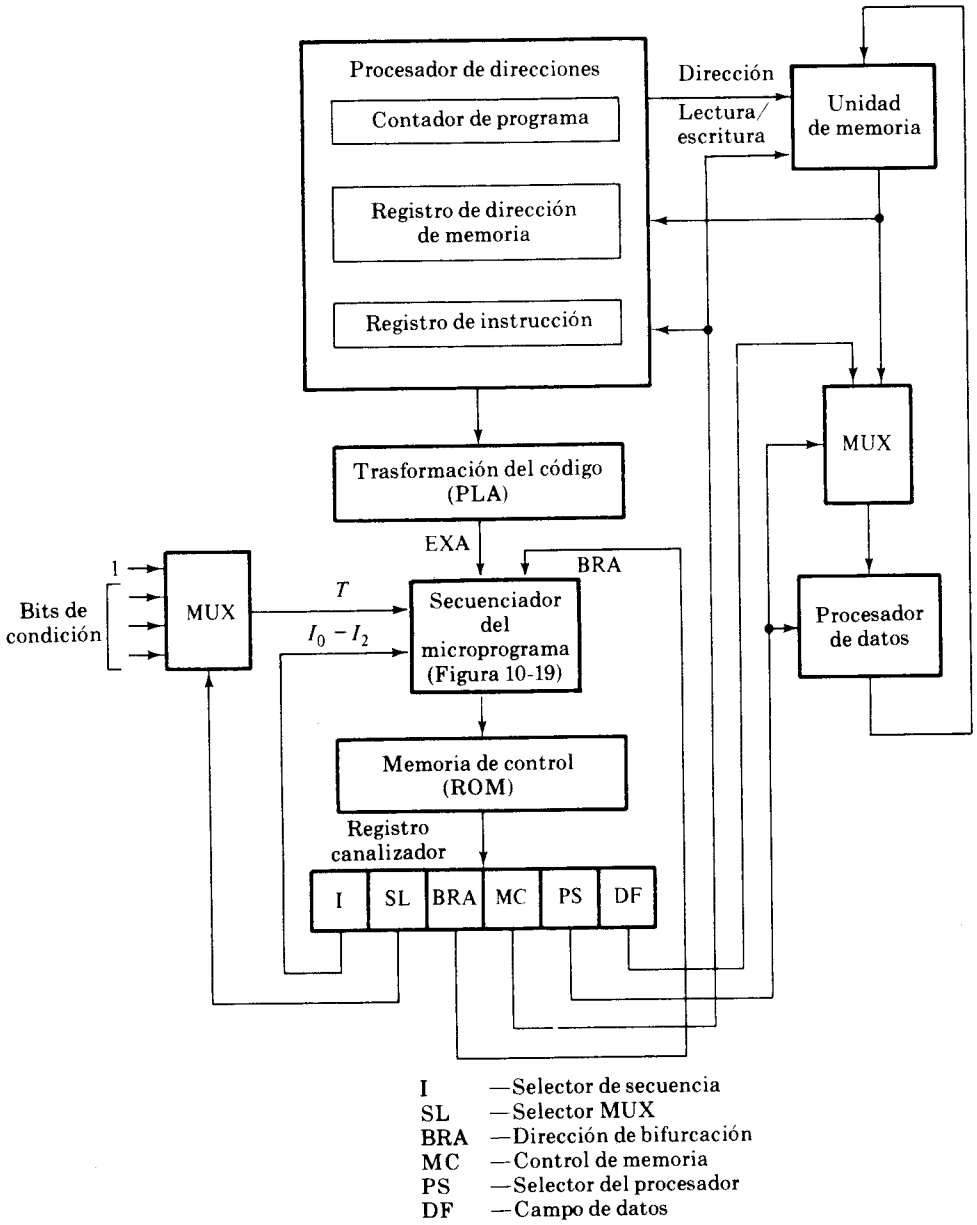


Figura 10-21 Organización del computador microprogramado

ción extraída de la memoria durante un ciclo de entrega o envío (fetch) pasa al registro de instrucción. Los bits del código instrucción en el registro de instrucción especifican una macrooperación para la memoria de control. A menudo se necesita una trasformación de código para convertir los bits del código de operación de una instrucción a la dirección de comienzo para la memoria de control. Este código de trasformación constituye una función de planimetría y puede ser configurada con una ROM o un PLA. El concepto de planimetría presenta flexibilidad para agregar instrucciones o macroinstrucciones para la memoria de control cuando se presente la necesidad. La dirección generada por la función *planimétrica* de trasformación del código se aplica a la entrada de la dirección externa (*EXA*) del secuenciador.

La unidad de control del microprograma consiste del secuenciador de la Figura 10-19, una memoria de control para almacenar microinstrucciones, un multiplexor y un registro canalizador (pipeline register). El multiplexor selecciona uno de los muchos bits de condición y lo aplica a la entrada (prueba) T del secuenciador. Una de las entradas del multiplexor es siempre 1 para presentar una operación de información incondicional. El registro canalizador no es siempre necesario ya que las salidas de la memoria de control van directamente a las entradas de control de varias unidades en el CPU. El registro canalizador, sin embargo, incrementa la velocidad de la operación de control. Este permite que se genere la siguiente dirección y que cambie la salida de la memoria de control mientras que la palabra de control corriente en el registro canalizador inicia las microoperaciones dadas por la presente microinstrucción.

Un posible formato de microinstrucción para la memoria de control se ilustra dentro del registro canalizador. El campo I consiste de tres bits y suministra la información de entrada al secuenciador. El campo SL selecciona el bit de condición para el multiplexor. El campo *BRA* es el campo de dirección de la microinstrucción que suministra una dirección de bifurcación (*BRA*) al secuenciador. Estos tres campos de la microinstrucción presentan información al secuenciador para determinar la siguiente dirección para la memoria de control. El secuenciador genera la siguiente dirección y la memoria de control lee la siguiente microinstrucción mientras que se vayan ejecutando las microoperaciones presentes en las otras unidades del CPU.

Los otros tres campos en la microinstrucción son para controlar las microoperaciones en la unidad del procesador y de memoria. El campo de la memoria de control (MC) controla al procesador de direcciones y las operaciones de lectura y escritura en la unidad de memoria. El campo de selección (PS) del procesador controla las operaciones en la unidad del procesador de datos. El último campo es el campo de datos (DF) usado para introducir constantes al microprocesador. El procedimiento para introducir datos al sistema a partir de la memoria de control es una técnica frecuentemente usada en muchos sistemas de microoperación. Las salidas del campo de datos pueden usarse para preparar registros de control e introducir datos en los registros del procesador. Por ejemplo, se puede agregar una constante del campo de datos al registro procesador para incrementar su contenido por un valor específico. Otro uso del campo de datos es poner un contador de secuencia a un valor constante. El contador de secuencia se usa para contar el número

de veces que se pasa por un bucle de microprograma (loop), como se requiere usualmente en la rutina de multiplicación o división.

Una vez que la configuración de los materiales del CPU microprogramado se establezcan, el diseñador lo puede usar para construir una cualquiera de las muchas configuraciones de computador posibles. Primero se formula la instrucción establecida para el computador y luego se escribe el microprograma para la memoria de control. Se puede cambiar el microprograma en la memoria de control si se desea un computador diferente con un conjunto diferente de instrucciones. No se necesitan cambios en los materiales si cambian las especificaciones del computador, ya que el cambio se hace solamente en la memoria de control ROM. Esto implica quitar la presente ROM de su base para cambiarla por otra con un microprograma diferente.

La construcción de un CPU a partir de componentes LSI como se muestra en la Figura 10-21, presenta la libertad para definir el conjunto de instrucciones de un sistema de computador. Se debe tener en cuenta, sin embargo, que los circuitos integrados disponibles contienen un CPU completo dentro de una sola cápsula. Este tipo de CPU se llama *microprocesador*. En el caso de usar un microprocesador en vez de un CPU hecho bajo pedido se debe estar de acuerdo con el conjunto de instrucciones fijas del microprocesador escogido. En otras palabras un microprocesador es un CPU ya listo, con un conjunto fijo de instrucciones de computador. Un CPU microprogramado hecho bajo pedido es una unidad flexible que permite la formulación de instrucciones adecuadas para una aplicación particular. Los microprocesadores se discuten en el Capítulo 12.

REFERENCIAS

1. Mano, M. M., *Computer System Architecture*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1976.
2. Rhyne, V. T., *Fundamentals of Digital Systems Design*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1973.
3. Chu, Y., *Computer Organization and Microprogramming*. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1972.
4. Mick, J. R. y J. Brick, *Microprogramming Handbook*. Sunnyvale, Calif.: Advance Micro Devices, Inc., 1977.
5. *Bipolar Microcomputer Components Data Book*. Dallas, Texas: Texas Instruments, Inc., 1977.
6. Agrawala, A. K. y T. G. Rauscher, *Foundations of Microprogramming*. Nueva York: Academic Press, 1976.
7. *Signetics Field Programmable Logic Array: An Application Manual*. Sunnyvale, Calif.: Signetics Corp., 1977.
8. Clare, C. R., *Designing Logic Systems Using State Machines*. Nueva York: McGraw-Hill Book Co., 1973.
9. Alexandridis, N. A., "Bit-sliced Microprocessor Architecture", *Computer*, Vol. 11, No. 6, (junio 1978), págs. 56-80.

PROBLEMAS

10-1. (a) Muestre que el control del contador de anillo de la Figura 7-22(a) es un caso especial de un flip-flop por cada control de estado dibujado en la Figura 10-2. Indique cómo el último puede reducirse al primero. (b) Muestre que el contador y el control del decodificador de la Figura 7-22(b) es un caso especial de un registro de secuencia y control de decodificador como se dibuja en la Figura 10-3. Indique cómo el último puede reducirse al primero.

10-2. ¿Cuál es la diferencia entre un control de conexiones de materiales y el control del microprograma? ¿Cuáles son las ventajas y desventajas de cada método?

10-3. El sistema sumador-sustractor diseñado en la Sección 10-3 usa un ALU. Redibuje el diagrama de bloque de la Figura 10-8 sin usar un ALU. En vez de ello use el circuito sumador sustractor de la Figura 9-10 y un registro con las características de complementar, incrementar y cargar. Revise las salidas de control de la Figura 10-9(b).

10-4. Repase el flujograma de la Figura 10-7 para buscar si podría resultar un cero negativo al final del cómputo. Un cero negativo ocurre si $A = 0$ y $A_s = 1$.

10-5. Diseñe un sistema digital que sume y reste dos números binarios de punto fijo representados en la forma de signo-complemento de 2. Incluya una indicación de sobrecapacidad.

10-6. Revise el diagrama de estado de control de la Figura 10-9 para constatar si el valor de C_{out} se comprueba en el tiempo T_4 en vez de comprobar el valor de E en el tiempo T_5.

10-7. El número de estados para el control de la Figura 10-9 puede reducirse si la variable S se usa conjuntamente con q_a y q_s para determinar el estado siguiente después del estado inicial. También, el registro A puede ser complementado durante el estado mismo en que E se borra si E' se incluye en la función de control para la operación de complemento. Demuestre que un sistema sumador sustractor puede ser configurado con seis estados de control.

10-8. Derive las funciones de entrada para los flip-flops B_s, A_s y E de la Figura 10-8(a). Use flip-flops JK.

10-9. Diseñe el control especificado por el diagrama de estado de la Figura 10-15(a) por el método de un flip-flop por estado. Dibuje el diagrama lógico usando compuertas y cuatro flip-flops D.

10-10. Obtenga una segunda lista para el microprograma binario de la Tabla 10-3, usando 00 para los bits de ROM 13 y 14 cada vez que se incrementa incondicionalmente el CAR.

10-11. Diseñe el circuito de entrada de la Figura 10-10 remplazando las compuertas AND asociadas con q_a y q_s con un multiplexor dual de 2 a 1 con entrada de habilitación.

10-12. La unidad de control de microprograma de la Figura 10-10 conjuntamente con el procesador de datos asociado de la Figura 10-8(a), se usan para sumar y restar dos números binarios en representación de signo-complemento de 2. Los bits de signo residen en la posición del bit de la externa izquierda de los registros A y B. Como los signos se incluyen con A y B, no hay necesidad de A_s y B_s y la variable S. En cambio, permítase que S sea el flip-flop que almacena el arrastre C_n que va en la posición del bit de signo, de la manera que E almacena al arrastre $C_{n+1} = C_{\text{out}}$ que sale de la posición del bit de signo. Su-

ponga que las variables y y z usan las dos señales de control que ponen a uno o a cero el flip-flop de sobrecapacidad V. Si ocurre una sobrecapacidad, V es puesta a uno con la variable del control y. Si no ocurre una sobrecapacidad, se borra V con la variable de control z.

(a) Escriba el microprograma en la forma simbólica.

(b) Liste la tabla de verdad de la ROM en binario.

10-13. Dé una microiñstrucción en forma binaria para la memoria de control de la Figura 10-11 que mantenga el sistema en un bucle de no operación siempre que la dirección externa sea igual a la dirección donde se localiza la microinstrucción en la memoria. Los valores que van al registro de condición no son importantes.

10-14. Escriba un microprograma en la forma simbólica para el sistema de la Figura 10-11 que comprueba el signo del número almacenado en el registro $R1$. El número está en la presentación de signo-complemento de 2. Si el número es positivo, se divide por 2. Si es negativo se multiplica por 2. Si ocurre una sobrecapacidad, $R1$ se lleva a 0.

10-15. Escriba un microprograma que compare dos números binarios sin signo almacenados en $R1$ y $R2$. El registro que contiene el número menor se borra entonces. Si los dos números son iguales, se borran ambos registros. Use el sistema de microprograma de la Figura 10-11.

10-16. El procesador de la Figura 9-16 se usa para multiplicar dos números binarios sin signo. El multiplicando está en $R1$, el multiplicador en $R3$ y el producto se forma en $R2$ y $R3$. El registro $R4$ almacena un número binario igual al número de bits en el multiplicador. Describa el algoritmo en la forma de flujograma.

10-17. Liste el contenido de los registros E, A, Q y P (Figura 10-16) después de cada pulso de reloj durante el proceso de multiplicación de las dos magnitudes 10111 (multiplicando) y 10011 (multiplicador).

10-18. El diagrama de estado de control de la Figura 10-15(a) no usa la variable Q_1 como una condición de transición de estado. En vez de ello, Q_1 se usa como parte de la función de control en la lista de registros de trasferencia. Diseñe el control de manera que Q_1 aparezca como una condición en el diagrama de estado y que se elimine de la lista de funciones de control. Muestre que para este caso el diagrama de estado debe tener al menos cinco estados.

10-19. Determine el tiempo que se toma procesar la operación de multiplicación en el sistema digital descrito en la Figura 10-15. Asuma que el registro Q tiene k bits y que el intervalo entre dos pulsos de reloj es t segundos.

10-20. Diseñe la lógica de control de la Figura 10-16 usando dos flip-flops T y un decodificador.

10-21. Cambie el registro P de la Figura 10-16 a un contador creciente con carga en paralelo. La entrada T_2 no incrementa el registro P. ¿Cuál es el valor inicial que debe cargarse a P en el tiempo T_1?

10-22. Pruebe que la multiplicación de dos números de n dígitos en cualquier base r da un poroducto de no más de $2n$ dígitos de longitud. Muestre que esta declaración implica que no puede ocurrir sobrecapacidad en el multiplicador diseñado en la Sección 10-6.

10-23. Diseñe el control especificado en la Figura 10-9 por el método de registro de secuencia y decodificador. Use los tres flip-flops JK, G_3, G_2 y G_1.

10-24. Diseñe el control especificado en la Figura 10-9 usando un registro de secuencia y un PLA. Liste la tabla de programa del PLA.

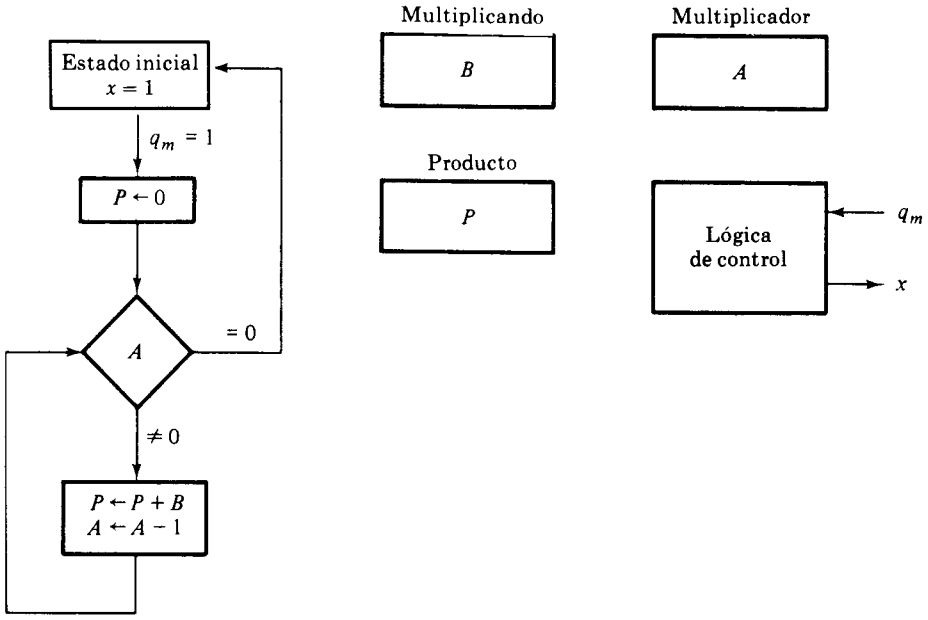


Figura P10-25 Multiplicación por sumas sucesivas

10-25. La configuración del registro y el flujograma del sistema digital que multiplica dos números binarios sin signo por el método de la adición repetida se muestra en la Figura P10-25.

(a) Convénzase usted mismo que el sistema multiplica el contenido de A y B y coloca el producto en el registro P.

(b) Sea $A = 0100$ y $B = 0011$. Pasando por los pasos del flujograma, demuestre que el sistema regresa a su estado incial, con el registro P conteniendo el producto 1100.

(c) Dibuje un diagrama de estado para el control y liste las trasferencias de registro que van a ejecutarse en cada estado de control.

(d) Dibuje el diagrama de bloque de la parte del procesador de datos.

(e) Diseñe el control por el método de un flip-flop por estado.

10-26. Las siguientes operaciones de trasferencia entre registros especifican un control del tipo de cuatro estados de registro de secuencia y decodificador. G es un registro de secuencia de 2 bits y T_0, T_1, T_2 y T_3 son las salidas del decodificador.

$$\begin{aligned} xT_0&: \quad G \leftarrow G + 1 \\ yT_0&: \quad G \leftarrow 10 \\ zT_0&: \quad G \leftarrow 11 \\ T_1 + T_2 + T_3&: \quad G \leftarrow G + 1 \end{aligned}$$

(a) Dibuje el diagrama de estado del control.

(b) Diseñe el registro de secuencia con flip-flops JK.

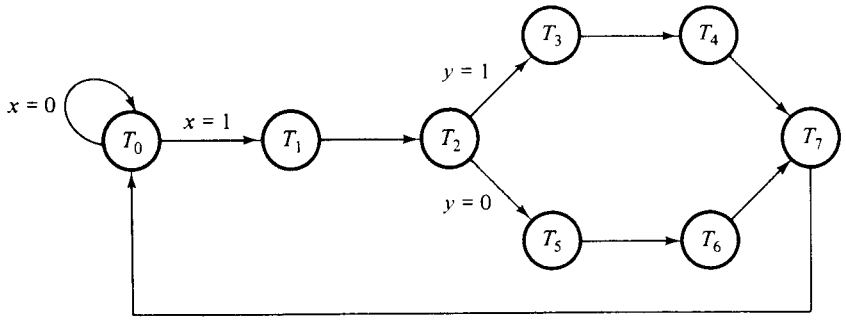


Figura P10-27 Diagrama de estado de control para el Problema 10-27

10-27. Una unidad de control tiene dos entradas x y y y ocho estados. El diagrama de estado de control se muestra en la Figura P10-27.

(a) Diseñe el control usando 8 flip-flops D.

(b) Diseñe el control usando un registro, un decodificador y un PLA.

10-28. El diagrama de estado de una unidad de control se muestra en la Figura P10-28. Tiene cuatro estados y dos entradas x y y. Diseñe el control por el método de registro de secuencia y decodificador con los dos flip-flops JK, G_2 y G_1.

(a) Use las salidas del decodificador como condiciones de estados presentes.

(b) Use las salidas de los flip-flops como condiciones de estados presentes. Compare los dos resultados y comente las ventajas y desventajas en cada caso.

10-29. El registro canalizador de la Figura 10-21 tiene una salida adicional demarcada P para controlar la polaridad de la entrada T en el secuenciador. Cuando $P = 0$ el valor del bit de condición seleccionado por SL se aplica a la entrada T. Cuando $P = 1$, el complemento del bit de condición seleccionado se aplica a T.

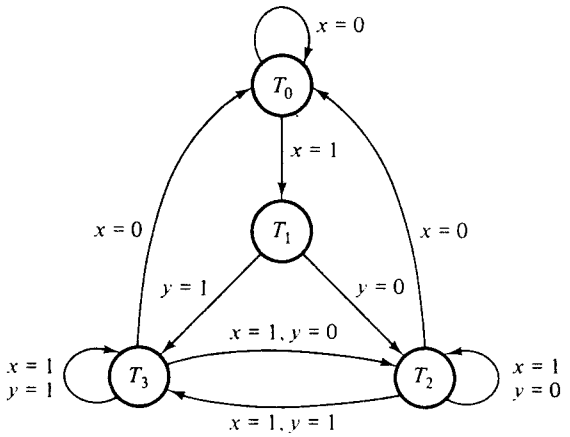


Figura P10-28 Diagrama de estado de control para el Problema 10-28

(a) ¿Qué se logra por medio del control de polaridad P?

(b) Diseñe el circuito que debe ser colocado entre el multiplexor seleccionado por SL y la entrada de prueba T.

10-30. El computador microprogramado de la Figura 10-21 tiene un registro de control de dirección (CAR) dentro del secuenciador y el registro canalizador (PLR) en la salida de la memoria de control. La velocidad de operación puede mejorarse si se usa solamente un registro. Compare la velocidad de operación comparando las demoras de propagación encontradas cuando se usa el sistema:

(a) Un CAR sin un PLR.

(b) Un PLR sin un CAR.

Diseño de computadores

11

11-1 INTRODUCCION

Este capítulo presenta un computador digital pequeño para propósitos generales a partir de sus especificaciones funcionales para culminar con su diseño. Aunque el computador es pequeño, está muy lejos de ser útil. Su finalidad es muy limitada cuando se compara con sistemas comerciales electrónicos de procesamiento de datos, aunque incluye especificaciones funcionales suficientes para demostrar el proceso de diseño. Es adecuado para construir en el laboratorio con CI y el producto terminado puede ser un sistema útil capaz de procesar datos digitales.*

El computador consiste de una unidad procesadora central, una unidad de memoria y una unidad teleimpresora de entrada-salida. El diseño lógico de la unidad procesadora central será derivado aquí. Las otras dos unidades se asumen como elementos disponibles con características externas conocidas.

El diseño de los materiales de un computador digital puede derivarse en tres fases interrelacionadas: diseño del sistema, diseño lógico y diseño del circuito. El diseño del sistema versa sobre las especificaciones y propiedades generales del sistema. Esta tarea incluye el establecimiento de un objetivo de diseño y filosofía de diseño, la formulación de las instrucciones del computador, y la investigación de su factibilidad económica. Las especificaciones de la estructura del computador son traducidas por el diseñador lógico para lograr la configuración de los materiales del sistema. El diseño del circuito especifica los componentes de los diferentes circuitos lógicos, circuitos de memoria, equipo electromecánico y suministro de potencia. El diseño de la parte de los materiales del computador es enormemente influenciado por el sistema de programación (software) el cual normalmente se desarrolla al tiempo y constituye una parte integral del sistema de computador total.

El diseño del computador digital es una tarea complicada. No se debe esperar el cubrimiento de todos los aspectos del diseño en un solo capítulo.

*Las instrucciones para el computador son un subconjunto de las instrucciones en el computador PDP-8.

Aquí interesa el sistema y el diseño lógico de un computador digital pequeño cuyas especificaciones se formulan de una manera arbitraria para poder establecer una configuración mínima para una máquina muy pequeña, aunque práctica. El procedimiento enunciado en este capítulo puede ser útil en el diseño lógico de sistemas más complicados.

El proceso del diseño se divide en seis fases:

1. La descomposición del computador digital en registros que especifican la configuración general del sistema.
2. La especificación de las instrucciones del computador.
3. La formulación de los circuitos de tiempo y de control.
4. La lista de operaciones de trasferencia entre registros necesaria para ejecutar todas las instrucciones de computador.
5. El diseño de la sección del computador.
6. El diseño de la sección del control.

El proceso de diseño se lleva a cabo por medio de las listas tabuladas que sumarizan las especificaciones y operaciones en forma compacta. La sección del procesador se define por medio del diagrama de bloque que consiste de registros y multiplexores. Se asume que el lector tiene información suficiente para remplazar los bloques en el diagrama con circuitos MSI. La sección de control se diseña para cada uno de los tres métodos enunciados en el Capítulo 10.

11-2 CONFIGURACION DEL SISTEMA

La configuración del computador se muestra en la Figura 11-1. Cada bloque representa un registro, excepto por la unidad de memoria, el generador del reloj maestro y la lógica de control. Esta configuración se supone que satisface la estructura del sistema final. En una situación práctica, el diseñador comienza con una configuración tentativa de un sistema y la modifica constantemente durante el proceso de diseño. El nombre de cada registro se escribe dentro del bloque, conjuntamente con la designación simbólica en paréntesis.

El generador del reloj maestro es una fuente de pulsos de reloj común, por lo general un oscilador, el cual genera un tren periódico de pulsos. A estos pulsos se les da mayor capacidad por medio de amplificadores y se distribuyen por todo el sistema. Cada pulso debe llegar a cada flip-flop y registro al mismo tiempo. El llevar los retardos a la misma fase puede necesitarse indirectamente de manera que la diferencia en retardos de trasmisión es uniforme por todas partes. La frecuencia de los pulsos es una función de la velocidad con la cual opera el sistema. Se asume una frecuencia de 1 megaciclo, la cual da un pulso cada milisegundo. Esta frecuencia de pulso se escoge para tener un número redondo y evitar problemas de los retardos de propagación en los circuitos.

La unidad de memoria tiene una capacidad de 4.096 palabras de 16 bits cada una. Esta capacidad es suficientemente larga para un proceso signifi-

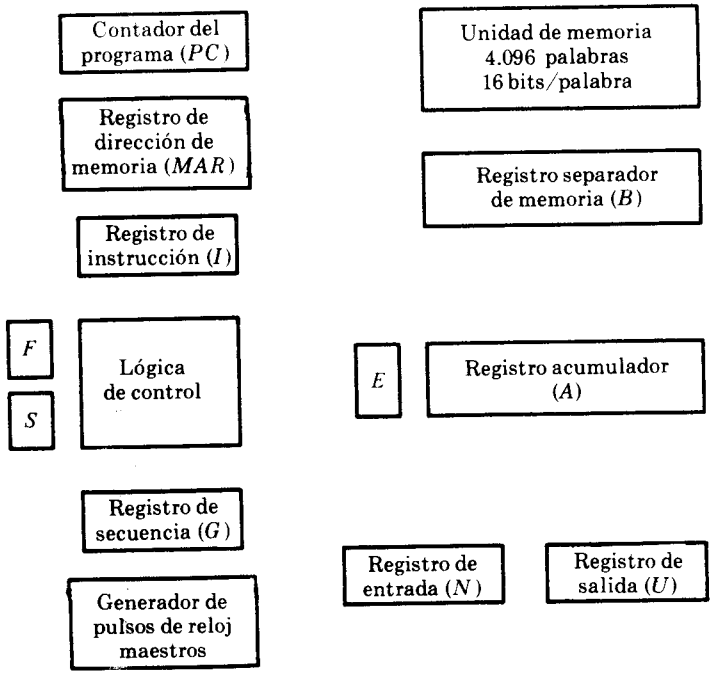


Figura 11-1 Diagrama de bloque del computador digital

Tabla 11-1 Lista de registros para el computador

Designación simbólica	Nombre	Número de bits	Función
A	Registro acumulador	16	Registro procesador
B	Registro separador de memoria	16	Retiene el contenido de la palabra de memoria
PC	Contador de programa	12	Retiene la dirección de la siguiente instrucción
MAR	Registro de dirección de memoria	12	Retiene la dirección de la palabra de memoria
I	Registro de instrucción	4	Retiene el código de operación corriente
E	Flip-flop de extensión	1	Extensión del acumulador
F	Flip-flop de búsqueda	1	Controla búsqueda y ejecuta ciclos
S	Flip-flop de comienzo-parada	1	Detiene y comienza el computador
G	Registro de secuencia	2	Entrega señales de tiempo
N	Registro de entrada	9	Retiene información del dispositivo de entrada
U	Registro de salida	9	Retiene información del dispositivo de salida

cativo. Se puede usar un tamaño menor si se desea construir un computador en el laboratorio bajo restricciones económicas. Son necesarios doce bits de una instrucción para especificar la dirección de un operando, el cual deja cuatro bits para la parte de la instrucción. El tiempo de acceso de la memoria se asume que es menor que 1 milisegundo de manera que la palabra puede leerse o escribirse durante el intervalo entre dos pulsos de reloj.

La parte del computador digital que se va a diseñar se descompone en subunidades de registro. Los siguientes parágrafos explican por qué cada registro es necesario y qué función realiza. Una lista de registros y una breve descripción de sus funciones se presenta en la Tabla 11-1. Los registros que almacenan palabras de memoria son de 16 bits de longitud. Aquellos que almacenan una dirección son de 12 bits de longitud. Otros registros tienen un número diferente de bits, dependiendo de su función.

Registro de dirección de memoria y registro separador de memoria

El registro de dirección de memoria, *MAR*, se usa para direccionar lugares de memoria específicos. El *MAR* se carga del contador del programa (*PC*) cuando una instrucción se va a leer de la memoria y de los 12 bits menos significativos del registro *B*, cuando un operando se va a leer de la memoria. El registro separador de memoria *B* almacena la palabra leída o escrita en la memoria. La parte de operación de una palabra de instrucción colocada en *B*, se trasfiere al registro *I* y la parte de la dirección se deja en el registro *B* para ser trasferida al *MAR*. Una palabra de operando colocada en el registro *B* se hace accesible para operación con el registro *A*. La palabra que va a ser almacenada en la memoria debe ser cargada al registro *B* antes de iniciar una operación de escritura.

Contador de programa

El contador de programa *PC* almacena la dirección de la siguiente instrucción para ser leída de la memoria. Este registro pasa por una secuencia de conteo paso a paso y causa que el computador lea instrucciones sucesivas almacenadas previamente en la memoria. Cuando el programa llama una trasferencia a otro lugar con el fin de evitar la siguiente instrucción en secuencia, se modifica el *PC* al mismo tiempo, causando que el programa continúe a partir de un lugar de memoria que está fuera de la secuencia de conteo. Para leer una instrucción, el contenido del *PC* se trasfiere al *MAR* y se inicia la operación de lectura. El contador del programa se incrementa siempre en 1 mientras que la operación de escritura de memoria lee la instrucción presente. Por tanto, la dirección de la siguiente instrucción, mayor una unidad que la que está siendo ejecutada en el procesador, está siempre disponible en el *PC*.

Registro acumulador

El registro acumulador *A* es un registro procesador que opera con datos previamente almacenados en la memoria. Este registro se usa para ejecutar

la mayoría de instrucciones y para aceptar datos del dispositivo de entrada o trasferir datos al dispositivo de salida. El registro *A* conjuntamente con el registro *B* componen el cuerpo de la unidad procesadora del computador. Aunque la mayoría de los sistemas procesadores de datos incluyen más registros para la unidad procesadora, se ha escogido incluir solamente un acumulador, para no complicar el diseño. Es posible configurar solamente la operación de suma con un simple acumulador con elemento aritmético. Otras operaciones aritméticas tales como sustracción, multiplicación y división pueden ser configuradas con una secuencia de instrucciones que forman una subrutina.

Registro de instrucción

El registro de instrucción *I* retiene los bits del código de operación de la instrucción corriente. Este registro tiene solamente cuatro bits ya que el código de operación de las instrucciones es de cuatro bits de longitud. Los bits del código de operación se trasfieren al registro *I* a partir del registro *B*, mientras que la parte de dirección de la instrucción se deja en *B*. La parte del código de operación se debe sacar del registro *B* ya que una lectura de un operando de la memoria al registro destruirá la instrucción retenida previamente. La parte de operación de la instrucción es necesaria para que el control determine qué se le va a hacer al operando que se ha leído.

Registro de secuencia

El registro de secuencia *G* es un contador que produce las señales de tiempo para el computador. El registro *G* es decodificado para suministrar cuatro variables de tiempo para la unidad de control. Las variables de tiempo conjuntamente con otras variables de control, producen las funciones de control que inician todas las microoperaciones del computador.

Flip-flops *E*, *F* y *S*

Cada uno de estos flip-flops se consideran un registro de un bit. El flip-flop *E* es una extensión del registro *A*. Se usa durante las operaciones de desplazamiento, recibe el arrastre final durante la suma y además es un flip-flop que puede simplificar las operaciones de procesamiento de datos del computador. El flip-flop *F* distingue entre los ciclos de búsqueda (fetch) y de ejecución. Cuando *F* es 0, la palabra leída de la memoria se trata como una instrucción. Cuando *F* es 1, la palabra se trata como un operando. *S* es un flip-flop de parada e inicio que puede borrarse por medio del control del programa y manejarse manualmente. Cuando *S* es 1, el computador trabaja de acuerdo a una secuencia determinada por el programa almacenado en la memoria. Cuando *S* es 0, el computador detiene su operación.

Registros de entrada y salida

El dispositivo de entrada-salida (I/O) no se muestra en el diagrama de bloque de la Figura 11-1. Se asume como una unidad de teleimpresora con

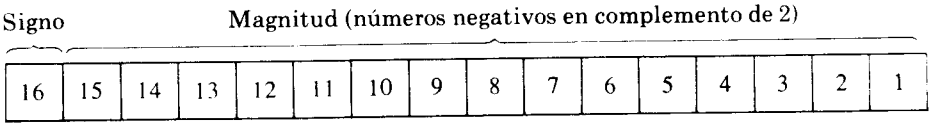
un teclado y un impresor. La teleimpresora envía y recibe información en serie. Cada paquete de información tiene ocho bits de un código alfanumérico. La información en serie de un teclado se desplaza al registro de entrada. La información en serie de un impresor se almacena en el registro de salida. Estos dos registros se comunican con la teleimpresora en serie y con el registro acumulador en paralelo.

El registro de entrada N consiste de nueve bits. Los bits 1 a 8 almacenan información de entrada alfanumérica; el bit 9 es un bit de control llamado *indicador* de entrada (flag). El bit indicador se pone a uno cuando está disponible un nuevo caracter de un dispositivo de entrada y se pone a cero cuando el caracter es aceptado por el computador. El bit indicador es necesario para sincronizar la baja velocidad a la cual opera el dispositivo de entrada comparada con la gran velocidad de los circuitos del computador. El proceso de trasferencia de información es como sigue. Inicialmente se borra el bit indicador en N_9. Cuando se oprime una tecla del teclado, el código de 8 bits se desplaza al registro de entrada ($N_1 - N_8$). Una vez que se termine la operación de desplazamiento, el bit indicador en N_9 se pone a 1. El computador comprueba el bit indicador; si éste es 1, el código de caracter del registro N se trasfiere en paralelo al registro A y el bit indicador se borra. Una vez se haya borrado el registro indicador, se puede desplazar el nuevo caracter al registro N al oprimir otra tecla.

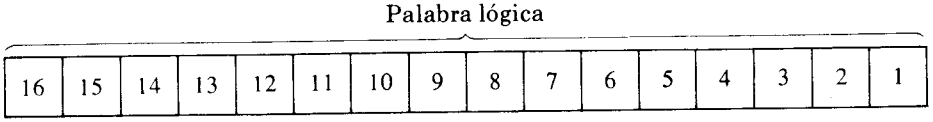
El registro de salida U trabaja de manera similar pero la dirección del flujo de información se invierte. Inicialmente, el indicador de salida en U_9 se pone a 1. El computador comprueba el bit indicador; si éste es 1, un código de caracter del registro A se trasfiere en paralelo al registro de salida ($U_1 - U_8$) y el bit indicador U_9 se borra a 0. El dispositivo de salida acepta la información codificada e imprime el caracter correspondiente; cuando se complete la operación se pone a 1 el bit indicador. El computador no carga un nuevo caracter al registro de salida, cuando el indicador es 0, porque esta condición indica que el dispositivo de salida está en proceso de imprimir el caracter previo.

11-3 INSTRUCCIONES DE COMPUTADOR

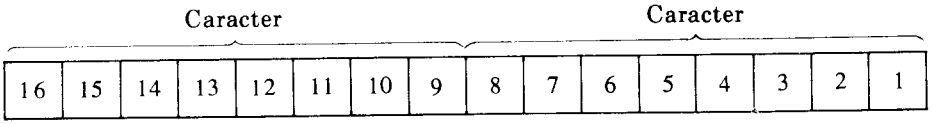
El número de instrucciones disponibles en un computador y su eficiencia en resolver el problema entre manos, es una buena indicación de lo bien que el diseñador del sistema previó la aplicación que se requería de la máquina. Los sistemas de cómputo mediano y de gran escala, pueden tener cientos de instrucciones, mientras que la mayoría de computadores pequeños limitan la lista a menos de 100. Las instrucciones se deben escoger con cuidado para imprimir las características suficientes del sistema con el fin de resolver un amplio rango de problemas de procesamiento de datos. Los requerimientos mínimos de tal lista deben incluir una capacidad para almacenar y cargar palabras de la memoria, un conjunto suficiente de operaciones aritméticas y lógicas, algunas propiedades de modificación de direcciones, bifurcación incondicional, bifurcación bajo condiciones de prueba, propiedades de manipulación de registro e instrucciones I/O. La lista de instrucciones escogidas para el computador pretende ser la mínima requerida para un procesador de datos práctico pero restringido.



(a) Operando aritmético



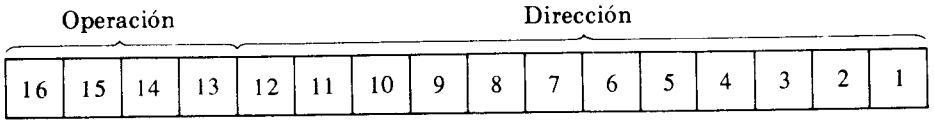
(b) Operando lógico



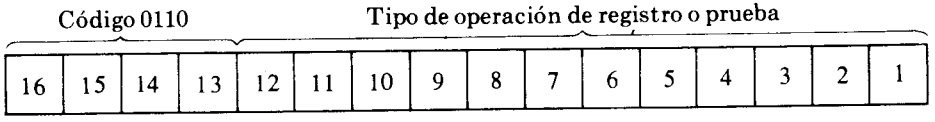
(c) Datos de entrada/salida

Figura 11-2 Formatos de datos

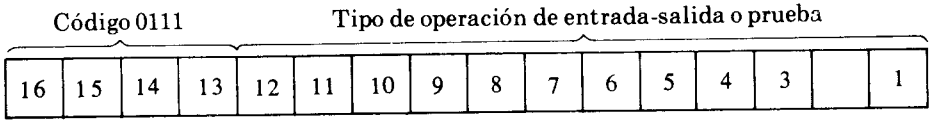
La formulación del conjunto de instrucciones para el computador va mano a mano con la formulación de los formatos para datos y palabras de instrucciones. Una palabra de memoria consiste de 16 bits. Una palabra puede representar una unidad de datos o una instrucción. Los formatos de palabras de datos se muestran en la Figura 11-2. Los datos para las operaciones aritméticas se representan por un número binario de 15 bits con un signo en la posición del bit décimo sexto. Se asume que los números están en su equivalente de complemento de 2. Las operaciones lógicas se realizan



(a) Instrucción de referencia de memoria



(b) Instrucción de referencia de registro



(c) Instrucción de entrada/salida

Figura 11-3 Formatos de instrucción

con bits individuales de la palabra, con el bit 16 tratado como cualquier otro bit. Cuando el computador se comunica con un dispositivo I/O, la información trasferida se considera compuesta de caracteres alfanuméricos de 8 bits. Dos de estos caracteres pueden acomodarse en una palabra de computador.

Los formatos de palabras de instrucción se muestran en la Figura 11-3. La parte operativa de la instrucción contiene cuatro bits; el significado de los 12 bits restantes depende del código de operación encontrado. Una instrucción de *referencia de memoria* usa los 12 bits restantes para especificar una instrucción. Una instrucción de *referencia de registro* implica una operación o prueba con el registro A o el E. No se necesita un operando de la memoria; por tanto, los 12 bits menos significativos son usados para especificar la operación o la prueba que se va a ejecutar. Una instrucción de referencia de registro se reconoce por el código 0110 en la parte de operación. Similarmente una instrucción de *entrada-salida* no necesita una referencia para la memoria y se reconoce por el código de operación 0111. Los 12 bits restantes se usan para especificar el dispositivo particular y el tipo de operación o prueba realizada.

Solamente cuatro bits de la instrucción están disponibles para el código de operación. Parecería que el computador está restringido a un máximo de 16 operaciones diferentes. Sin embargo, como las instrucciones de referencia de registro y de entrada-salida usan los 12 bits restantes como parte del código de operación, el número total de instrucciones puede exceder de 16. De hecho, el número total de instrucciones escogido para el computador es 22.

De las 16 diferentes operaciones que pueden ser formuladas con cuatro bits, solamente ocho pueden ser utilizadas por el computador, porque el bit de la extrema izquierda de todas las instrucciones (bit 16) es siempre 0. Esto deja abierta la posibilidad de agregar nuevas instrucciones y aumentar la capacidad del computador, si se desea.

Las seis instrucciones de referencia de memoria para el computador se listan en la Tabla 11-2. El diseño simbólico es una palabra de tres letras y

Tabla 11-2 Instrucciones de referencia de memoria

Símbolo	Código hexadecimal	Descripción	Función
AND	0 m*	AND a A	$A \leftarrow A \wedge M$*
ADD	1 m	Sumar a A	$A \leftarrow A + M$, $E \leftarrow$ Arrastre
STO	2 m	Almacenar en A	$M \leftarrow A$
ISZ	3 m	Incrementar y omitir si es cero	$M \leftarrow M + 1$, si $(M + 1 = 0)$ entonces $(PC \leftarrow PC + 1)$
BSB	4 m	Bifurcar a una subrutina	$M \leftarrow PC + 5000$, $PC \leftarrow m + 1$
BUN	5 m	Bifurcar incondicionalmente	$PC \leftarrow m$

*m es la parte de dirección de la instrucción. M es la palabra de memoria direccionada por m.

representa una abreviación que va a ser usada por programadores y usuarios cuando se escriben programas simbólicos para el computador. El código hexadecimal listado es un número hexadecimal equivalente al código binario adoptado por el código de operación. Una instrucción de referencia de memoria usa un dígito hexadecimal (4 bits) para el código de operación; los tres dígitos hexadecimales restantes (12 bits) de la instrucción representan una dirección designada por la letra *m*. Cada instrucción tiene una breve descripción en palabras y se especifica más precisamente en la columna de función con una proposición de macrooperación. Una clarificación de cada instrucción se da a continuación, conjuntamente con una explicación de su uso.

AND de *A*

Esta es una operación lógica que ejecuta una operación AND a los pares de bits correspondientes en *A*, con la palabra *M* de memoria especificada por la parte de dirección de la instrucción. El resultado de la operación se deja en el registro *A*, remplazando su contenido anterior. Cualquier computador debe tener un conjunto básico de operaciones lógicas para el manipuleo de los datos no numéricos. Las operaciones lógicas más comunes que se encuentran en las instrucciones del computador son el AND el OR, el OR-exclusivo y el complemento. Aquí se usa solamente el AND y el complemento. Este último se incluye como instrucción de referencia de registro. Estas dos operaciones lógicas constituyen un conjunto mínimo del cual se pueden derivar todas las demás operaciones lógicas porque el AND y el complemento juntos, forman la operación NAND. En la Sección 4-7 se observó que ésta es una operación universal de la cual se puede obtener cualquier otra operación lógica.

SUMAR a *A* (ADD)

Esta instrucción agrega el contenido de la palabra *M* de memoria, especificada por la parte de dirección de la instrucción al contenido presente del registro *A*. La suma se hace asumiendo que los números negativos estén en su forma de complemento de 2. Esto requiere que el bit de signo se agregue de la misma forma que se agregan todos los demás bits. El arrastre de salida proveniente de la posición del bit de signo se trasfiere al flip-flop *E*. Esta instrucción, conjuntamente con las instrucciones de referencia de registro, es suficiente para escribir programas para configurar todas las demás operaciones aritméticas. La sustracción se logra complementando e incrementando el sustraendo. La multiplicación se logra sumando y desplazando. El incremento y desplazamiento son instrucciones de referencia de registro.

La instrucción ADD (sumar) debe ser usada para cargar una palabra de la memoria al registro *A*. Esto se hace borrando el registro *A* con la instrucción CLA de referencia de registro (definida en la Tabla 11-3). La palabra requerida se descarga de la memoria agregándole al registro *A* previamente borrado.

ALMACENAR en *A* (STORE)

Esta instrucción almacena el contenido del registro *A* en la palabra de memoria especificada por la dirección de instrucción. Las tres instrucciones de referencia de memoria se usan para manipular los datos, entre la palabra de memoria y el registro *A*. Las tres instrucciones siguientes son instrucciones de control que causan un cambio en la secuencia normal del programa.

Incrementar y omitir en caso de cero (ISZ)

La instrucción de incrementar y omitir es útil para la modificación de dirección y para contar el número de veces que se ejecuta un bucle de programa. Un número negativo almacenado previamente en la memoria en la dirección *m* se lee por medio de la instrucción ISZ. Este número se incrementa en 1 y se almacena de nuevo a la memoria. La instrucción siguiente es omitida si después de ser incrementada el número llega a cero. Así, al final del bucle de programa, se coloca una instrucción ISZ seguida de una instrucción de bifurcación (BUN) que incondicionalmente vaya al comienzo del bucle del programa. Si el número almacenado no llega a cero, el programa regresa de nuevo a ejecutar el bucle. Si éste llega a cero, la siguiente instrucción (BUN) es omitida y el programa continúa ejecutando instrucciones después del bucle del programa.

Bifurcación incondicional (BUN)

Esta instrucción trasfiere el control incondicionalmente a la instrucción en el lugar especificado por la parte *m* de la dirección. Recuérdese que el contador del programa retiene la dirección de la siguiente instrucción que se va a leer y ejecutar. Normalmente el *PC* se incrementa para dar la dirección de la siguiente instrucción en secuencia. El programador tiene la prerrogativa de especificar cualquier otra instrucción fuera de la secuencia usando la instrucción BUN. Esta instrucción le dice al computador que tome la parte *m* de la dirección y la trasfiera al *PC*. La dirección de la siguiente instrucción que se va a ejecutar estará ahora en el *PC* y es aquella que antes era la parte de la dirección de la instrucción BUN.

La instrucción BUN está listada con las instrucciones de referencia de memoria porque necesita una parte *m* de dirección. Sin embargo, no necesita una referencia de la memoria para tener acceso a una palabra de memoria (designada por el símbolo *M*), como se requiere por otras instrucciones de referencia de memoria.

Bifurcar a la subrutina (BSB)

Esta instrucción es útil para la bifurcación a la parte de una subrutina de programa. Cuando se ejecuta, ésta almacena la dirección de la siguiente instrucción en secuencia, la cual está almacenada al presente en el *PC* (llamada *dirección de retorno*), en la palabra de memoria especificada por la parte de dirección de la instrucción. También almacena el código de opera-

ción del BUN (hexadecimal 5) en el mismo lugar de la memoria. El contenido de la parte m de la dirección más 1 se trasfiere al PC para comenzar la ejecución del programa de la subrutina en esta parte específica. Una vez se haya ejecutado la subrutina, el control se trasfiere al programa de llamado por medio de una instrucción BUN colocada al final de la subrutina.

El proceso de bifurcación a la subrutina y el regreso al programa de llamado se muestra en la Figura 11-4 por medio de un ejemplo numérico específico. El programa de llamado está ahora en la posición 32. El programa de subrutina comienza en la posición 65. La instrucción BSB causa una trasferencia a la subrutina y la última instrucción en la rutina causa una bifurcación de regreso hacia la posición 33 en el programa de llamado. El ejemplo numérico en la Figura 11-4 muestra una instrucción BSB en la posición 32 con una parte m de la dirección igual al binario 64. Mientras que se ejecuta esta instrucción el PC almacena la dirección de la siguiente instrucción en secuencia, la cual es 33. La instrucción BSB realiza la macrooperación (ver Tabla 11-2):

$$M \leftarrow PC + 5000, \quad PC \leftarrow m + 1$$

El contenido del PC más el hexadecimal 5000 (código para el BUN) son trasferidos a la posición 64. Esta trasferencia produce una instrucción BUN 33. La parte de la dirección de la instrucción se incrementa y se coloca en el PC. El PC almacena el binario equivalente a 65 de manera que el computador comienza a ejecutar la subrutina en esta posición. La última instrucción en la subrutina es BUN 64. Cuando esta instrucción se ejecute, el control se trasfiere a la instrucción en la posición 64. Pero en la posición 64, hay una instrucción que se bifurca de regreso a la dirección 33. La di-

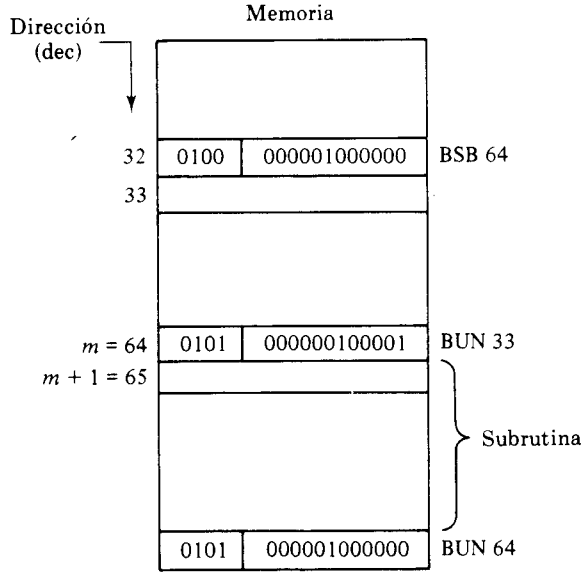


Figura 11-4 Demostración de una instrucción de bifurcación a la subrutina

rección almacenada en la posición 64 por la instrucción BSB tendrá la dirección de regreso propia, no importando dónde esté la instrucción BSB localizada. De esta manera, el regreso a la subrutina es siempre a una posición un lugar mayor que la posición de la instrucción BSB. Nótese que el número de la dirección de la instrucción BUN localizado al final de la subrutina debe ser siempre igual al número de la dirección donde se almacena temporalmente la dirección de regreso, el cual es 64 en este caso.

Instrucciones de referencia de registro

Las 12 instrucciones de referencia de registro para el computador se listan en la Tabla 11-3. Cada instrucción de referencia de registro tiene un código de operación 0110 (hexadecimal 6) y contiene un solo 1 en uno de los 12 bits restantes de la instrucción. Estas instrucciones se especifican con cuatro dígitos hexadecimales, los cuales representan todos los 16 bits de una palabra de instrucción. Las primeras siete instrucciones realizan una operación sobre los registros A y E y se explican por sí solas. Las siguientes cuatro instrucciones de omisión (skip) se usan para control del programa condicionado por ciertos bits de condición. Para omitir la siguiente instrucción el PC se incrementa en 1 de nuevo. El primer incremento ocurre cuando se lee la instrucción actual. De esta manera la siguiente instrucción leída de la memoria está dos posiciones más arriba de la posición de la presente instrucción (omisión).

Los bits de condición para las instrucciones de omisión son el bit de signo en A, el cual está en el flip-flop A_{16} y la condición de cero para A ó E. Si la condición de status designado está presente, se omite la siguiente instrucción en secuencia; de otra manera el computador continúa a partir de la siguiente instrucción en secuencia, debido a que el PC no se incrementa.

Tabla 11-3 Instrucciones de referencia de registro

Símbolo	Código hexadecimal	Descripción	Función
CLA	6800	Borrar A	$A \leftarrow 0$
CLE	6400	Borrar E	$E \leftarrow 0$
CMA	6200	Complementar A	$A \leftarrow \overline{A}$
CME	6100	Complementar E	$E \leftarrow \overline{E}$
SHR	6080	Desplazar a la derecha A y E	$A \leftarrow \text{shr } A, A_{16} \leftarrow E, E \leftarrow A_1$
SHL	6040	Desplazar a la izquierda A y E	$A \leftarrow \text{shl } A, A_1 \leftarrow E, E \leftarrow A_{16}$
INC	6020	Incrementar A	$A \leftarrow A + 1$
SPA	6010	Omitir con A positivo	Si $(A_{16} = 0)$ entonces $(PC \leftarrow PC + 1)$
SNA	6008	Omitir con A negativo	Si $(A_{16} = 1)$ entonces $(PC \leftarrow PC + 1)$
SZA	6004	Omitir con A cero	Si $(A = 0)$ entonces $(PC \leftarrow PC + 1)$
SZE	6002	Omitir con E cero	Si $(E = 0)$ entonces $(PC \leftarrow PC + 1)$
HLT	6001	Detener el computador	$S \leftarrow 0$

La instrucción de detención (halt) se coloca por lo general al final de un programa si se desea detener el computador. Su ejecución borra el flip-flop de parada y comienzo para evitar operaciones posteriores.

Instrucciones de entrada-salida

El computador tiene cuatro instrucciones de entrada-salida que se listan en la Tabla 11-4. Estas instrucciones tienen un código de operación 0111 (hexadecimal 7) y cada una contiene un 1 en solamente uno de los 12 bits restantes de la palabra de instrucción. Las instrucciones de entrada-salida se especifican con cuatro dígitos hexadecimales comenzando con 7.

Tabla 11-4 Instrucciones de entrada-salida

Símbolo	Código hexadecimal	Descripción	Función
SKI	7800	Omitir con el indicador de entrada	Si ($N_9 = 1$) entonces ($PC \leftarrow PC + 1$)
INP	7400	Introducir a A	$A_{1-8} \leftarrow N_{1-8}$, $N_9 \leftarrow 0$
SKO	7200	Omitir con el indicador de salida	Si ($U_9 = 1$) entonces ($PC \leftarrow PC + 1$)
OUT	7100	Sacar de A	$U_{1-8} \leftarrow A_{1-8}$, $U_9 \leftarrow 0$

La instrucción INP trasfiere el caracter de entrada de N hasta A y borra también el indicador de entrada en N_9. La instrucción OUT trasfiere un código de caracter de 8 bits desde A hasta el registro de entrada y también borra el indicador de salida en U_9. Las dos instrucciones de omisión comprueban los indicadores de condición correspondientes y causan una omisión de la siguiente instrucción si el bit indicador es 1. La instrucción que se omite es una instrucción BUN. La instrucción BUN no se omite si el bit indicador es 0; ésta causa una bifurcación de regreso a la instrucción de omisión para comprobar de nuevo el indicador. Si el bit indicador es 1 se omite la instrucción BUN y se ejecuta la operación de entrada o salida. Así, el computador permanece en un bucle de dos instrucciones (omisión en indicación y bifurcación de regreso a la instrucción anterior) hasta que el bit indicador sea puesto a uno mediante un dispositivo externo. La siguiente instrucción en secuencia debe ser una instrucción de entrada o salida.

11-4 SINCRONIZACION DE TIEMPO Y CONTROL

Todas las operaciones del computador están sincronizadas por un generador de tiempo maestro cuyos pulsos de reloj se aplican a todos los flip-flops del sistema. Además, está disponible cierto número de variables de tiempo en la unidad de control para darle secuencia a la operación en el orden adecuado. Esas variables de tiempo se designan como t_0, t_1, t_2 y t_3 y se muestran en la Figura 11-5. Los pulsos de reloj ocurren una vez cada microsegundo (μs). Cada variable de tiempo es de 1 μs de duración y ocurre una vez cada 4 μs. Se asume que el disparo de los flip-flops ocurre

durante el flanco negativo de los pulsos de reloj. Se puede controlar el pulso de reloj específico que dispara el registro, aplicando una de las variables de tiempo al terminal de entrada de habilitación de un registro dado. Las variables de tiempo se repiten continuamente de manera que t_0 aparezca después de t_3. Cuatro variables de tiempo son suficientes para la ejecución de cualquier instrucción en el computador que consideramos aquí. En otras situaciones podría ser necesario emplear un número diferente de variables de tiempo.

Se asume que el tiempo de acceso de memoria es menor que 1 μs. Una operación de lectura o escritura de memoria puede iniciarse con una de las variables de tiempo cuando ésta se ponga alta. La operación de memoria se completará en el momento en que llegue el pulso siguiente de reloj.

El computador digital opera con pasos discretos controlados por las señales de tiempo. Una instrucción és leída de la memoria y ejecutada en los registros por medio de una secuencia de microoperaciones. Cuando el control recibe una instrucción, éste genera las funciones de control adecuadas para las microoperaciones requeridas. En la Figura 11-6 se muestra un diagrama de bloque de la lógica de control. Una instrucción que se lee de la memoria se coloca en el registro B separador de la memoria. La instrucción tiene un código de operación de 4 bits, designado por el símbolo OP. Si ésta es una instrucción de referencia de memoria tendrá una parte de dirección designada por el símbolo AD. El código de operación se trasfiere siempre al registro de instrucción I. El código de operación en I se decodifica en ocho salidas $q_0 - q_7$, siendo el número suscrito igual al código hexadecimal para la operación. El registro G es un contador de 2 bits que cuenta continuamente los pulsos de reloj, durante el tiempo en que el flip-flop S, de comienzo-parada, esté puesto a uno. Las salidas del registro G se decodifican en cuatro variables de tiempo $t_0 - t_3$. El flip-flop F distingue entre los ciclos de búsqueda y de ejecución. Otras condiciones de status son necesarias para determinar la secuencia de control. Las salidas de los circuitos de lógica de control indican todas las microoperaciones para el computador. El diagrama de bloque de la lógica de control es útil para visualizar la unidad de control del computador, cuando se derivan las operaciones de trasferencia entre registros durante el proceso de diseño de la lógica.

La red de lógica de control es un circuito combinacional que consiste de una conexión aleatoria de compuertas. Su configuración constituye un control de elementos interconectados. Se verá en la Sección 11-7 que la parte de control del computador puede configurarse con arreglos lógicos programables. La configuración del PLA remplaza la red lógica de control, como también los decodificadores de operación y sincronización. En la Sección 11-7 se muestra que el control puede ser configurado parcialmente con una unidad de microprograma. La configuración de control del microprograma remplazará la red de lógica de control, los dos decodificadores y los registros I y G.

11-5 EJECUCION DE INSTRUCCIONES

Hasta el momento se ha considerado el diseño del sistema del computador. Se ha especificado la configuración del registro, el conjunto de instruccio-

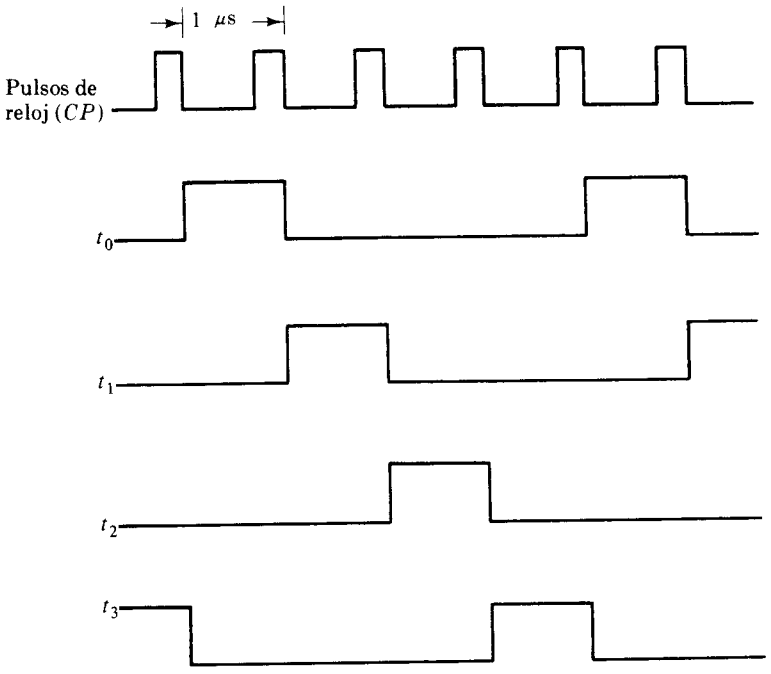


Figura 11-5 Señales de tiempo del computador

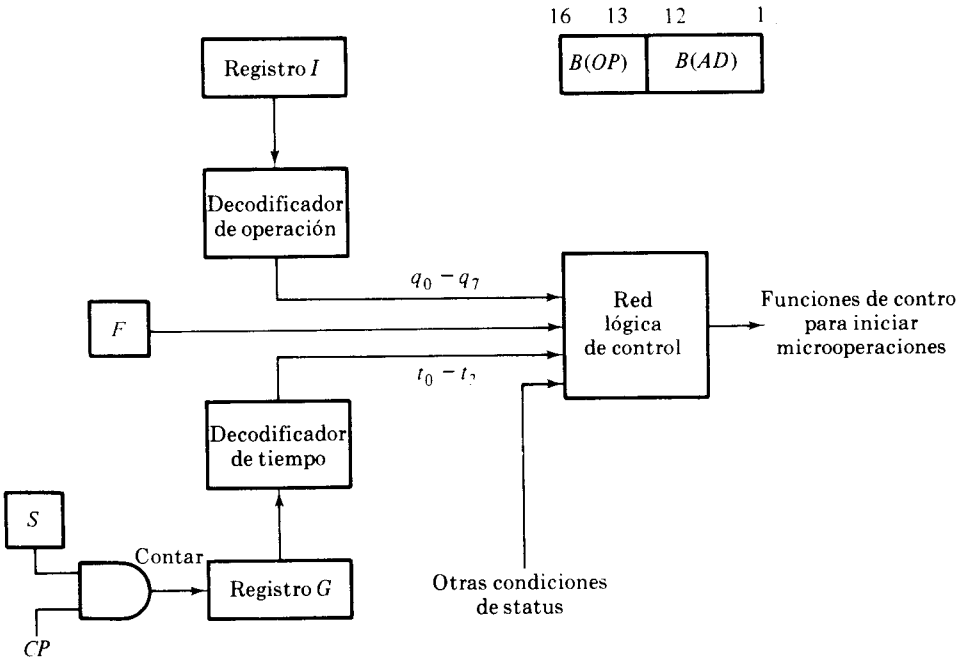


Figura 11-6 Diagrama de bloque de la lógica de control

nes del computador, una secuencia de tiempo y la configuración de la unidad de control. En esta sección, se comienza con la fase de diseño del computador. El primer paso es especificar las microoperaciones conjuntamente con las funciones de control necesarias para ejecutar cada instrucción de máquina.

Las operaciones de trasferencia entre registros describen de una forma concisa el proceso de trasferencia de información dentro de los registros del computador. Cada proposición de la descripción consiste de una función de control, seguida de una coma, seguida de una o más microoperaciones con notación simbólica. La función de control es una función de Boole cuyas variables son las señales de tiempo $t_0 - t_3$, la operación decodificada $q_0 - q_7$ y ciertas condiciones de los bits de condición. Las microoperaciones son especificadas de acuerdo con la notación simbólica definida en el método de trasferencia entre registros.

Una vez que se active el interruptor de comienzo, la secuencia del computador sigue un patrón básico. Una instrucción cuya dirección está en el PC se lee de la memoria. Su parte de operación se trasfiere al registro I y el PC se incrementa en 1 para prepararla para la dirección de la siguiente instrucción. Si la instrucción es del tipo referencia de memoria, podría ser necesario accesar de nuevo la memoria para leer el operando. Así, las palabras leídas de la memoria al registro B pueden ser instrucciones o datos. El flip-flop F se usa para distinguir entre los dos. Cuando $F = 0$, la palabra leída de la memoria se interpreta como una instrucción y se dice que el computador está en el ciclo de búsqueda de la instrucción. Cuando $F = 1$, la palabra leída de la memoria se toma como un operando y se dice que el computador está en el ciclo de ejecución.

Ciclo de búsqueda (fetch)

Una instrucción se lee de la memoria durante el ciclo de búsqueda. Las relaciones de trasferencia entre registros que especifican este proceso son:

$$\begin{aligned} F't_0: &\quad MAR \leftarrow PC \\ F't_1: &\quad B \leftarrow M,\ PC \leftarrow PC + 1 \\ F't_2: &\quad I \leftarrow B(OP) \end{aligned}$$

Cuando $F = 0$, las señales de tiempo t_0, t_1 y t_2 inician una secuencia de operaciones que trasfieren el contenido del PC al MAR, inician una lectura de memoria, incrementan el PC y trasfieren el código de operación de la instrucción al registro I. Todas las microoperaciones se ejecutan cuando la función de control es de lógica 1 y cuando ocurre un pulso de reloj. Las microoperaciones en los registros y la trasferencia de la palabra de memoria a B son ejecutadas durante el flanco negativo del pulso de reloj. Esto ocurre justamente antes del momento en que la variable de tiempo especificada va a cero.

El código de operación en el registro I se decodifica en el tiempo t_3. El siguiente paso depende del valor de q_i, $i = 0, 1, \ldots, 7$, que produce un 1 a la salida del decodificador. Si la salida decodificada es una instrucción de

referencia de memoria, se puede necesitar un operando. Si no, se puede ejecutar la instrucción durante el tiempo t_3.

La instrucción BUN y las instrucciones de referencia de registros e instrucciones de entrada-salida no necesitan un segundo acceso de la memoria. El computador tiene que pasar a un ciclo de ejecución para accesar de nuevo la memoria, cuando se encuentra con un código de operación 0, 1, 2, 3, ó 4. Esta condición se detecta a partir del decodificador de operación, el cual causa una trasferencia al ciclo de ejecución, poniendo F a 1:

$$F'(q_0 + q_1 + q_2 + q_3 + q_4)t_3: \quad F \leftarrow 1$$

Las operaciones comunes de trasferencia entre registros durante el ciclo de búsqueda se listan en la Tabla 11-5.

Tabla 11-5 Operaciones de trasferencia entre registros durante el ciclo de búsqueda

$F't_0$:	$MAR \leftarrow PC$	Trasferir la dirección de instrucción
$F't_1$:	$B \leftarrow M$, $PC \leftarrow PC + 1$	Leer la instrucción, incrementar PC
$F't_2$:	$I \leftarrow B(OP)$	Trasferir código de operación
$F'(q_0 + q_1 + q_2 + q_3 + q_4)t_3$:	$F \leftarrow 1$	Ir a ejecutar ciclo
q_5t_3:	$PC \leftarrow B(AD)$	Bifurcar incondicionalmente (BUN)
q_6t_3:	Ver Tabla 11-8	Instrucción de referencia de registros
q_7t_3:	Ver Tabla 11-9	Instrucción de entrada-salida

Las instrucciones BUN tienen un código de operación 5 y su correspondiente salida del decodificador de operación es q_5. Esta instrucción no necesita un operando de la memoria, aunque esté listado como una instrucción de referencia de memoria. Esta especifica que la siguiente instrucción se tome de la posición dada por la parte m de la dirección. La parte de dirección de la instrucción está en $B(AD)$ en el tiempo t_3 del ciclo de búsqueda. La instrucción puede ejecutarse durante el ciclo de búsqueda en ese tiempo:

$$q_5t_3: \quad PC \leftarrow B(AD)$$

No hay necesidad de incluir F en la función de control porque el único tiempo en que q_5 puede ser 1 es durante el ciclo de búsqueda. La microoperación que ejecuta la instrucción especifica una trasferencia de los bits 1 hasta 12 del registro B al PC. La siguiente variable de tiempo después de t_3 es siempre t_0. Como F permanece en 0 para esta instrucción, el computador regresa al comienzo del ciclo de búsqueda para leer la instrucción dada por el PC.

Las instrucciones de referencia de registros son reconocidas a partir de la salida del decodificador q_6 y las instrucciones de entrada-salida de q_7. Como estas instrucciones requieren solamente una microoperación más para su ejecución, ellas se pueden determinar en el tiempo t_3 durante el ciclo de búsqueda. Este hecho se indica en la Tabla 11-5. Las microoperaciones específicas se listan en tablas posteriores.

Ciclo de ejecución

El flip-flop F es igual a 1 durante el ciclo de ejecución. Las cuatro variables de tiempo que ocurren durante el ciclo realizan las microoperaciones para ejecutar una de las instrucciones de referencia de memoria. La instrucción que se va a ejecutar se especifica por medio de la variable q_i, $i = 0, 1, 2, 3, 4$ disponible para el decodificador de operación. La parte de dirección de la instrucción estará en los bits 1 a 12 del registro B, simbolizado $B(AD)$, al final del ciclo de búsqueda. Esta dirección se trasfiere al MAR al comienzo del ciclo de ejecución para servir como la dirección de memoria para la palabra de memoria subsecuente:

$$Ft_0: \quad MAR \leftarrow B(AD)$$

Las instrucciones que necesitan un operando de la memoria son el AND (q_0), ADD (q_1) e ISZ (q_3). Las otras dos instrucciones STO (q_2) y BSB (q_4) almacenan un valor en la memoria y son ejecutadas durante la siguiente operación de lectura de memoria:

$$F(q_0 + q_1 + q_3)t_1: \quad B \leftarrow M$$

La instrucción decodificada particular se ejecuta con las variables de tiempo t_2 y t_3. En el tiempo t_3 se borra el flip-flop F para que el computador regrese al ciclo de búsqueda:

$$Ft_3: \quad F \leftarrow 0$$

La siguiente variable de tiempo después de t_3 es t_0. Pero como ahora F es igual a 0, por tanto la siguiente función de control es $F't_0$. Esta es la primera función de control en el ciclo de búsqueda. Así, después de ejecutar la instrucción corriente, el control regresa siempre al ciclo de búsqueda para leer la siguiente instrucción cuya dirección está en el PC. Las operaciones comunes realizadas durante el ciclo de ejecución se listan en la Tabla 11-6.

Las cinco instrucciones de referencia de memoria y sus correspondientes operaciones de registro se listan en la Tabla 11-7. Estas instrucciones son ejecutadas cuando $F = 1$ y con las variables de tiempo t_2 y t_3. La operación decodificada q_i determina la instrucción particular que se ejecuta.

Tabla 11-6 Operaciones comunes para el ciclo de ejecución

Ft_0:	$MAR \leftarrow B(AD)$	Trasferir parte de la dirección
$F(q_0 + q_1 + q_3)t_1$:	$B \leftarrow M$	Leer el operando
$F(t_2 + t_3)$:	Ver Tabla 11-7	Ejecutar la instrucción de referencia de memoria
Ft_3:	$F \leftarrow 0$	Regresar al ciclo de búsqueda

Tabla 11-7 Ejecutar las instrucciones de referencia de memoria

AND	Fq_0t_3:	$A \leftarrow A \wedge B$	Microoperación AND
ADD	Fq_1t_3:	$A \leftarrow A + B$, $E \leftarrow$ Arrastre	Microoperación de suma
STO	Fq_2t_2:	$B \leftarrow A$	Trasferir A a B
	Fq_2t_3:	$M \leftarrow B$	Almacenar en la memoria
ISZ	Fq_3t_2:	$B \leftarrow B + 1$	Incrementar la palabra de memoria
	Fq_3t_3:	$M \leftarrow B$	Almacenar de nuevo en la memoria
	$Fq_3B_zt_3$:	$PC \leftarrow PC + 1$	Omitir si $B_z = 1$ ($B = 0$)
BSB	Fq_4t_2:	$B(AD) \leftarrow PC$, $B(OP) \leftarrow 0101$, $PC \leftarrow MAR$	Trasferir a la dirección de regreso, trasferir la dirección al PC
	Fq_4t_3:	$M \leftarrow B$, $PC \leftarrow PC + 1$	Almacenar la dirección de regreso, incrementar la dirección en el PC

Las instrucciones AND y ADD se ejecutan con las variables de tiempo t_3, aunque pueden usar las variables t_2 como remplazo. El operando de la memoria ha sido trasferido a B con la variable de tiempo t_1. La operación correspondiente puede ejecutarse ahora entre los registros B y A.

La instrucción STO especifica una trasferencia del contenido de A a la palabra de memoria cuyas direcciones fueron trasferidas al MAR con la variable de tiempo t_0. El contenido de A se trasfiere primero a B y la operación de escritura trasfiere el contenido de B a la palabra de memoria especificada por MAR:

$$Fq_2t_2: \quad B \leftarrow A$$

$$Fq_2t_3: \quad M \leftarrow B$$

La instrucción ISZ se ejecuta con las siguientes microoperaciones:

$$Fq_3t_2: \quad B \leftarrow B + 1$$

$$Fq_3t_3: \quad M \leftarrow B$$

$$Fq_3B_zt_3: \quad PC \leftarrow PC + 1 \qquad B_z = 1 \text{ if } B = 0$$

La palabra de la posición M fue colocada en B durante el tiempo t_1 (ver Tabla 11-6). El registro B se incrementa en el tiempo t_2 y el nuevo valor se almacena una vez más en la memoria. Durante todo este tiempo el MAR no cambia, de manera que especifica siempre la dirección de M. Recuérdese que una palabra de memoria no puede ser incrementada cuando esté localizada en la memoria. Debe ser trasferida a un registro procesador donde se pueda ejecutar el conteo. Mientras que el número incrementado se almacena en la memoria, se prueba su valor en B; si es 0, se incrementa el PC para causar una omisión de una instrucción. La variable B_z usada en la última proposición es una variable de detección de cero y es igual al binario 1 si el registro B contiene un número de solo ceros.

La instrucción BSB es la instrucción disponible más complicada en el computador. Una forma posible de ejecutar esta instrucción es como sigue:

$$Fq_4t_2: \quad B(AD) \leftarrow PC,\ B(OP) \leftarrow 0101,\ PC \leftarrow MAR$$
$$Fq_4t_3: \quad M \leftarrow B,\ PC \leftarrow PC + 1$$

La dirección de regreso disponible en el PC se trasfiere a la parte de dirección del registro B y el código 0101 (BUN) se trasfiere a la parte del código de operación del mismo registro. Recuérdese que el registro de dirección MAR contiene la parte de la dirección de la instrucción designada por m. La trasferencia del MAR al PC resulta de la trasferencia de m al PC. Todo lo anterior se hace durante la variable de tiempo t_2. La dirección de regreso se almacena en la memoria en el tiempo t_3. El PC se incrementa en este tiempo de manera que la instrucción que se lea durante el siguiente ciclo de búsqueda estará en la posición $m + 1$.

Instrucciones de referencia entre registros

Las microoperaciones de registro que ejecutan las instrucciones de referencia de registros se listan en la Tabla 11-8. Estas instrucciones se reconocen en el terminal de salida q_6 del decodificador de operación y son ejecutadas durante el tiempo t_3 del ciclo de búsqueda. Por conveniencia se define una nueva variable $r = q_6 t_3$ y se usa en todas las funciones de control de referencia entre registros. El resto de las funciones de control se determinan a partir de uno de sus bits en el registro B, mientras que el resto de la instrucción permanece almacenado en ese tiempo. Por ejemplo, la instrucción CLA tiene el código hexadecimal 6800, el cual corresponde al código binario 0110 1000 0000 0000. El código de operación se decodifica a partir del registro I y es igual a q_6. El bit 12 en el registro B es 1; de manera que la función de control que ejecuta esta instrucción es $q_6 t_3 B_{12} = rB_{12}$.

Las primeras siete instrucciones de referencia entre registros ejecutan las operaciones de borrado, complemento, desplazamiento e incremento en el registro A ó E. Las siguientes cuatro instrucciones son instrucciones de

Tabla 11-8 Ejecución de las instrucciones de referencia entre registros

	$r = q_6t_3$		
CLA	rB_{12}:	$A \leftarrow 0$	Borrar A
CLE	rB_{11}:	$E \leftarrow 0$	Borrar E
CMA	rB_{10}:	$A \leftarrow \overline{A}$	Complementar A
CME	rB_9:	$E \leftarrow \overline{E}$	Complementar E
SHR	rB_8:	$A \leftarrow \text{shr } A,\ A_{16} \leftarrow E,\ E \leftarrow A_1$	Desplazar a la derecha A y E
SHL	rB_7:	$A \leftarrow \text{shl } A,\ A_1 \leftarrow E,\ E \leftarrow A_{16}$	Desplazar a la izquierda A y E
INC	rB_6:	$A \leftarrow A + 1$	Incrementar A
SPA	$rB_5A'_{16}$:	$PC \leftarrow PC + 1$	Incrementar PC si A es positivo
SNA	rB_4A_{16}:	$PC \leftarrow PC + 1$	Incrementar PC si A es negativo
SZA	rB_3A_z:	$PC \leftarrow PC + 1$	Incrementar PC si A es cero
SZE	rB_2E':	$PC \leftarrow PC + 1$	Incrementar PC si E es cero
HLT	rB_1:	$S \leftarrow 0$	Borra el flip-flop de comienzo y parada

omisión ejecutadas solamente si se satisfacen las condiciones establecidas. La omisión de una instrucción se logra incrementando de nuevo PC además del incremento que se hace en el tiempo t_1 (ver Tabla 11-5). El estado del bit de condición para la omisión se convierte en parte de la función de control. Así, el acumulador es positivo si $A_{16} = 0$ y negativo si $A_{16} = 1$. El símbolo A_z es una variable binaria igual a 1 cuando el registro A contiene sólo ceros. E' es igual a 1 cuando el flip-flop E contiene 0.

La instrucción de parada (halt) borra el flip-flop de comienzo y parada S y detiene la secuencia de tiempo. El registro de secuencia G para de contar mientras que su valor sea 0. Esto causa que el computador esté latente con t_0 siempre a la salida del decodificador de tiempo. Como F es también 0, la función de control $F't_0$ es la única que se produce mientras que el computador esté inactivo. Esta función de control trasfiere el contenido del PC al MAR continuamente (ver Tabla 11-5). Esta trasferencia continua se puede tolerar cuando el computador hace una parada latente. Si esto es indeseable se pueden quitar los pulsos de reloj del MAR y de la misma manera prevenir que esta trasferencia ocurra cuando $S = 0$. El computador puede recomenzar cuando se active el interruptor de "comienzo", el cual pone a 1 el flip-flop S. Esto causa que los pulsos de reloj alcancen la secuencia del registro G y comiencen a producir las otras variables de tiempo.

Instrucciones de entrada-salida

Las microoperaciones de trasferencia entre registros que ejecutan las cuatro instrucciones de entrada-salida se listan en la Tabla 11-9. Estas instrucciones son reconocidas en el terminal de salida q_7 del decodificador de operación y se ejecutan durante el tiempo t_3. Se define una nueva variable $p = q_7 t_3$ y se usa en todas las funciones de control de entrada-salida. Las funciones de control para estas instrucciones contienen un solo bit del registro B, el cual es parte de la definición del código de instrucción. Las dos instrucciones de omisión dependen del estado de las condiciones de los bits indicadores N_9 y U_9.

Tabla 11-9 Ejecución de las instrucciones de entrada-salida

	$p = q_7 t_3$		
SKI	$pB_{12}N_9$:	$PC \leftarrow PC + 1$	Incrementar PC si el indicador de $N_9 = 1$
INP	pB_{11}:	$A_{1-8} \leftarrow N_{1-8}$, $N_9 \leftarrow 0$	Alimentar A, borrar indicador
SKO	$pB_{10}U_9$:	$PC \leftarrow PC + 1$	Incrementar PC si el indicador de salida $U_9 = 1$
OUT	pB_9:	$U_{1-8} \leftarrow A_{1-8}$, $U_9 \leftarrow 0$	Extraer de A, borrar indicador

11-6 DISEÑO DE LOS REGISTROS DE COMPUTADOR

El diseño de un sistema digital sincrónico sigue un procedimiento prescrito. A partir del conocimiento de las necesidades del sistema se formula una red de control y se obtiene una lista de operaciones de trasferencia entre registros del sistema. Una vez que se haya derivado esa lista, el resto del

diseño es directo. Algunas instalaciones utilizan técnicas de automatización para el diseño de computador para traducir las proposiciones de trasferencia entre registros a un diagrama de circuitos compuesto de circuitos integrados.

La Sección 11-5 especifica las proposiciones de trasferencia entre registros para el computador en cinco tablas separadas. Las entradas en las tablas consisten de funciones de control y microoperaciones. La lista de funciones de control presenta las funciones de Boole para las compuertas en la red lógica de control. La lista de microoperaciones da una indicación del tipo de registros que deben escogerse para el computador. Aunque estas tablas son suficientes para completar el diseño lógico del sistema, podría ser conveniente redistribuir la información en las tablas de una manera más conveniente durante el proceso de configuración actual.

Operaciones de registro

Para determinar el tipo de terminal de control que se debe tener en cada registro es necesario obtener una lista de microoperaciones que afecten cada registro separadamente. Esto puede lograrse repasando las tablas de la Sección 11-5 y escogiendo aquellas proposiciones que cambien el contenido de un registro en particular. Esto se aplica también a las operaciones de lectura y escritura en la unidad de memoria. Por ejemplo, una operación de lectura de memoria se simboliza con la microoperación:

$$B \leftarrow M$$

La proposición indica también que el contenido del registro B cambiará el valor. Esta proposición se encuentra dos veces en la lista de microoperaciones. En la Tabla 11-5, se encuentra con la función de control $F't_1$ y en la Tabla 11-6 con la función de control $F(q_0 + q_1 + q_3)t_1$. Como ambas funciones de control producen la misma operación, se pueden combinar con una OR para dar la proposición:

$$R = F't_1 + F(q_0 + q_1 + q_3)t_1: \quad B \leftarrow M$$

El símbolo R se usa por conveniencia para designar la operación de *lectura* con una sola variable de control de *Boole*. El símbolo igual después de R designa su igualdad con las funciones de control listadas.

Este proceso se repite para la operación de escritura en memoria y para todos los registros del computador. El resultado es como se muestra en la Tabla 11-10. A cada función de control listada en la tabla se le asigna un nombre de variable de control. Las variables de una sola letra no son necesarias, pero ayudan a acortar las expresiones algebraicas del control de entrada de los registros. En la mayoría de los casos se asigna a la variable de control una letra subíndice idéntica a la letra mayúscula reservada para simbolizar el registro correspondiente. Las variables de control comunes al mismo registro se distinguen por subíndices numéricos diferentes.

La Tabla 11-10 se deriva directamente de las Tablas 11-5 a 11-9. El registro al cual pertenece una microoperación se reconoce por la presencia de

un símbolo en el lado izquierdo de la flecha. Para reconocer las microoperaciones que pertenecen al registro A se repasan las operaciones listadas en las Tablas 11-5 hasta 11-9 y se escogen las que tienen una A como registro de destino. Las microoperaciones para los otros registros se obtienen de manera similar. Si la microoperación ocurre más de una vez, las funciones de control correspondientes se aplican a una OR para producir la función de control compuesta.

Las operaciones para el flip-flop E deben separarse de las operaciones para el registro A, aunque ellas se habían listado conjuntamente en las tablas anteriores. La operación de desplazamiento circular a la derecha, por ejemplo, se enuncia en la Tabla 11-8 como:

$$rB_8: \quad A \leftarrow \text{shr } A, A_{16} \leftarrow E, E \leftarrow A_1$$

Nótese que r es una variable igual a $q_6 t_3$ y rB_8 se le asigna una variable de control a_5. En la Tabla 11-10 bajo el registro A se tiene:

$$a_5 = rB_8: \quad A \leftarrow \text{shr } A, A_{16} \leftarrow E$$

el cual es parte de la operación de desplazamiento que cambia el contenido de A. Debajo del flip-flop E se tiene:

$$a_5 = rB_8: \quad E \leftarrow A_1$$

lo cual muestra la parte de la operación de desplazamiento que cambia el flip-flop E. Así, la variable a_5 de control del desplazamiento a la derecha, desplaza el contenido de A a la derecha y coloca el valor de E en el bit de la extrema izquierda de A. Trasfiere también el bit de la extrema derecha de A a E.

La secuencia del registro G no tiene ningunas microoperaciones listadas en las tablas previas. Este registro se muestra en la Figura 11-6 como un contador cuyos pulsos de reloj se habilitan por medio del flip-flop S de comienzo y parada. Esto se incluye en la Tabla 11-10 con la proposición:

$$S: \quad G \leftarrow G + 1$$

Diseño del computador

La lista de microoperaciones dadas en la Tabla 11-10 suministra la información necesaria para diseñar los registros del computador. Las operaciones que se van a realizar en cada registro se demuestran claramente en las proposiciones listadas. Por ejemplo, el contador del programa PC tiene tres microoperaciones:

$$c_1: \quad PC \leftarrow PC + 1$$

$$c_2: \quad PC \leftarrow B(AD)$$

$$b_3: \quad PC \leftarrow MAR$$

Tabla 11-10 Microoperaciones para los registros

Memoria de control		
$R = F't_1 + F(q_0 + q_1 + q_3)t_1$:	$B \leftarrow M$	Leer de memoria
$W = F(q_2 + q_3 + q_4)t_3$:	$M \leftarrow B$	Escribir en memoria
Registro A		
$a_1 = Fq_0t_3$:	$A \leftarrow A \wedge B$	AND
$a_2 = Fq_1t_3$:	$A \leftarrow A + B$	Sumar
$a_3 = rB_{12}$:	$A \leftarrow 0$	Borrar
$a_4 = rB_{10}$:	$A \leftarrow \overline{A}$	Complementar
$a_5 = rB_8$:	$A \leftarrow \text{shr } A, A_{16} \leftarrow E$	Desplazamiento a la derecha
$a_6 = rB_7$:	$A \leftarrow \text{shl } A, A_1 \leftarrow E$	Desplazamiento a la izquierda
$a_7 = rB_6$:	$A \leftarrow A + 1$	Incremento
$a_8 = pB_{11}$:	$A_{1-8} \leftarrow N_{1-8}$	Trasferencia
Registro B		
$b_1 = Fq_2t_2$:	$B \leftarrow A$	Trasferencia
$b_2 = Fq_3t_2$:	$B \leftarrow B + 1$	Incremento
$b_3 = Fq_4t_2$:	$B(AD) \leftarrow PC, B(OP) \leftarrow 0101$	Trasferencia
Registro PC		
$c_1 = F't_1$ $+(q_3B_z + q_4)Ft_3$ $+(B_5A'_{16} + B_4A_{16}$ $+ B_3A_z + B_2E')r$ $+(B_{12}N_9 + B_{10}U_9)p$:	$PC \leftarrow PC + 1$	Incremento
$c_2 = q_5t_3$:	$PC \leftarrow B(AD)$	Trasferencia
$b_3 = Fq_4t_2$:	$PC \leftarrow MAR$	Trasferencia
Registro MAR		
$d_1 = F't_0$:	$MAR \leftarrow PC$	Trasferencia
$d_2 = Ft_0$:	$MAR \leftarrow B(AD)$	Trasferencia
Registro I		
$i_1 = F't_2$:	$I \leftarrow B(OP)$	Trasferencia
Flip-flop E		
$e_1 = rB_{11}$:	$E \leftarrow 0$	Borrar
$e_2 = rB_9$:	$E \leftarrow \overline{E}$	Complementar
$a_2 = Fq_1t_3$:	$E \leftarrow$ arrastre	Trasferencia
$a_5 = rB_8$:	$E \leftarrow A_1$	Desplazamiento a la derecha
$a_6 = rB_7$:	$E \leftarrow A_{16}$	Desplazamiento a la izquierda
Flip-flop F		
$f_1 = F'(q_0 + q_1 + q_2$ $+ q_3 + q_4)t_3$:	$F \leftarrow 1$	Poner a 1
$f_2 = Ft_3$:	$F \leftarrow 0$	Borrar
Flip-flop S		
$s_1 = rB_1$:	$S \leftarrow 0$	Borrar
Registro G		
S:	$G \leftarrow G + 1$	Contar
Registro U		
$u_1 = pB_9$:	$U_{1-8} \leftarrow A_{1-8}, U_9 \leftarrow 0$	Trasferir
Registro N		
$a_8 = pB_{11}$:	$N_9 \leftarrow 0$	Borrar

Este registro debe tener condiciones de incremento y trasferencia. Se puede configurar por medio de un contador con carga en paralelo del tipo mostrado en la Figura 7-19. Como el *PC* recibe información de entrada de dos fuentes, éste requiere un multiplexor para seleccionar entre dos entradas, como se explica en asocio con la Figura 8-3. Los otros registros se diseñan de manera similar.

El diagrama de bloque que muestra los tipos de registros necesarios para el computador se da en la Figura 11-7. La unidad de memoria se incluye también para mostrar su conexión al procesador. La lógica de control presenta todas las variables de control de los registros. El diseño de la lógica de control se discute en la siguiente sección. Las variables de control que se generan en la unidad de control son aplicadas a los registros de la manera que se indica en el diagrama. Además de los registros, el procesador usa cuatro multiplexores para seleccionar de dos o más fuentes. Todos los registros y multiplexores son funciones MSI disponibles en circuitos integrados normales. Los tres flip-flops *E* y *F* y *S* y su correspondiente lógica combinacional debe diseñarse con compuertas SSI y con flip-flops.

Todos los registros en el computador excepto el registro *A* requieren terminales de entrada de control de carga, incremento o de carga e incremento juntos. Se puede escoger el uso de un contador MSI con carga en paralelo para todos los registros. De esta manera se podría tener un inventario de un circuito integrado normal para los registros. Un componente comercial posible es el CI tipo 74161. Este circuito MSI contiene un contador de 4 bits con una carga en paralelo y un terminal de entrada de borrado asincrónico. Los terminales de entrada de borrado de los registros pueden conectarse a un interruptor de puesta a cero maestro en el computador para borrar todos los registros asincrónicos antes de las operaciones con reloj. Los registros de 12 bits, el *PC* y el *MAR* necesitan tres CI y el registro *B* de 16 bits, cuatro CI. Los registros *I* y *G* pueden configurarse con un solo CI cada uno. El contador de 4 bits de CI puede convertirse a un contador de 2 bits para *G* por el método enunciado en la Sección 7-5 en asocio con la Figura 7-20.

El registro *A* es el más complicado porque realiza todas las tareas de procesamiento del computador. Este registro es un registro acumulador del tipo diseñado en la Sección 9-10 y puede usar la configuración mostrada en la Figura 9-22. Puede configurarse también con un registro de desplazamiento bidireccional con la carga en paralelo como se muestra en la Figura 7-9, conjuntamente con un ALU del tipo discutido en la Sección 9-6. Una mejor forma sería usar un circuito acumulador MSI tal como el CI tipo 74S281. Cuando se configura con un ALU o acumulador de CI, la unidad de control debe generar las variables de control correspondientes para seleccionar las microoperaciones requeridas en el ALU. Estas serán diferentes de las funciones de control definidas por la unidad de control en este diseño.

El registro de entrada *N* y el registro de salida *U* pueden ser parte de la interconexión normal de una teleimpresora. Los circuitos integrados que hacen interconexión con una unidad teleimpresora están disponibles comercialmente y se les llaman a menudo *trasmisores-receptores asincrónicos universales* (universal asynchronous receiver-transmitters abreviado

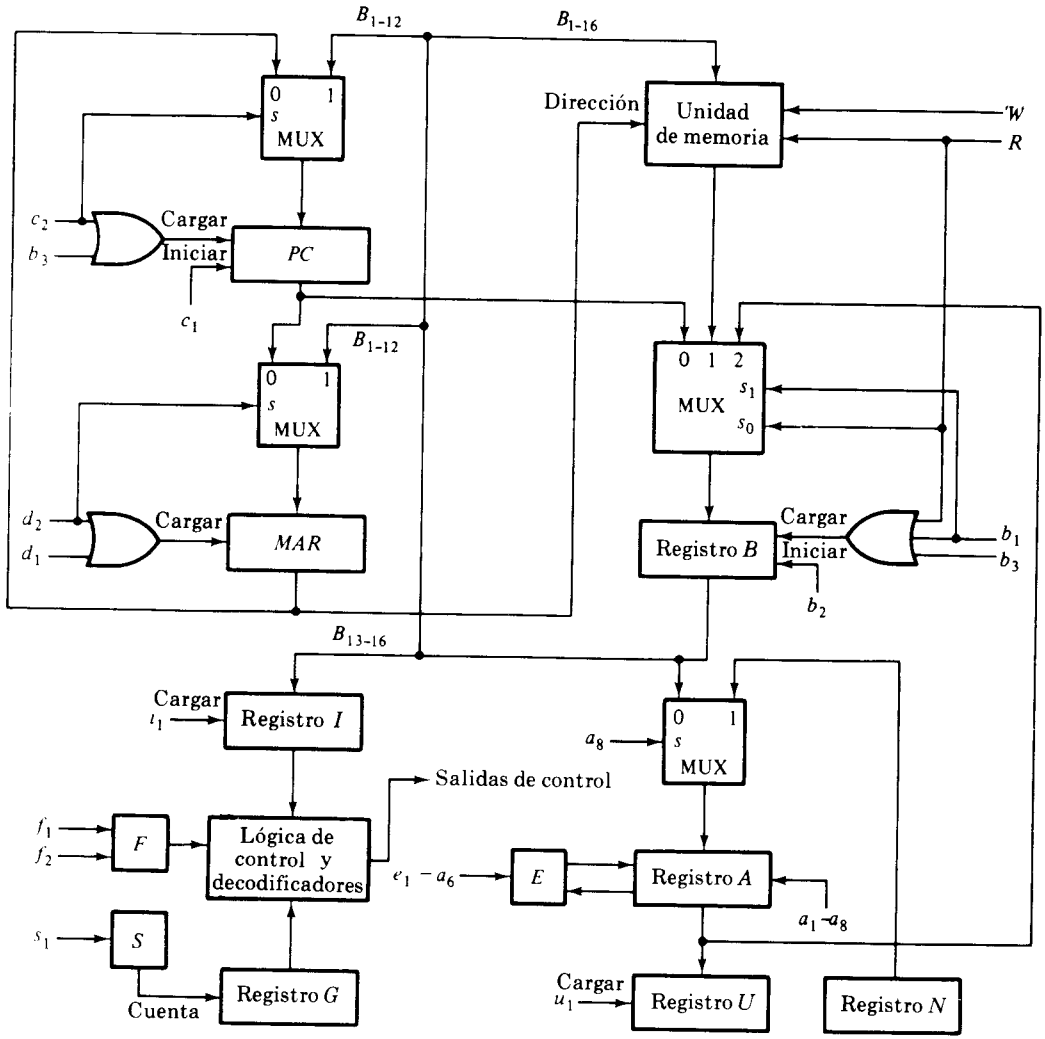


Figura 11-7 Diagrama de bloque detallado del computador

UART). Tal circuito integrado incluye un circuito de entrada y de salida dentro de la unidad conjuntamente con dos indicadores necesarios para sincronizar la trasferencia.

Tres de los multiplexores de la Figura 11-7 seleccionan entre dos fuentes de entrada. El terminal de entrada número 1 del MUX se selecciona cuando el terminal de entrada de selección marcado con una s es 1. Cuando $s = 0$ se selecciona el terminal de entrada número 0 del MUX. El multiplexor asociado con el registro B tiene tres fuentes de entrada. Las variables de selección s_1 y s_0 determinan la entrada seleccionada. Cuando ambas líneas de selección son 0, la entrada seleccionada proviene del PC. La señal R de lectura de memoria hace $s_0 = 1$ mientras que s_1 permanece en 0

(porque $b_1 = 0$ cuando $R = 1$). La entrada número 1 del MUX se selecciona mediante la entrada que proviene de la unidad de memoria, cuando $s_1 s_0 = 01$. De manera similar la variable de control b_1 produce la selección $s_1 s_0 = 10$ la cual causa la selección del contenido del registro A.

El computador entero mostrado en la Figura 11-7 puede encapsularse dentro de un solo CI para formar un *microcomputador*. Un microcomputador de CI típico normalmente tiene características nuevas en la sección del procesador, pero incluye una memoria más pequeña. La mayor parte de la memoria de un computador es comúnmente del tipo ROM. El diseño interno de la cápsula del microcomputador requiere que la lógica del computador se defina con un conjunto de funciones de Boole que especifican todas las compuertas y flip-flops en el sistema. Las funciones de Boole que configuran cada registro en el sistema pueden deducirse por el método presentado en la Sección 9-10 para el diseño de los registros en términos de funciones de Boole.

11-7 DISEÑO DEL CONTROL

La unidad de control del computador genera las variables de control para los registros y unidad de memoria. Hay 24 variables de control diferentes y la mayoría de ellas se listan en la Tabla 11-10 como funciones de control. En el Capítulo 10 se presentaron tres métodos para el diseño de la lógica de control: el control con componentes alambrados, el control PLA y el control del microprograma. La unidad de control del computador puede ser diseñada usando cualquiera de estos tres métodos.

Control con componentes alambrados

La organización de control presentada en la Figura 11-16 es esencialmente una organización con componentes alambrados por el método del registro de secuencia y decodificador. El registro de secuencia G en este caso es un contador y el decodificador de tiempo entrega cuatro estados de control para el sistema. Un segundo decodificador se usa para el código de operación almacenado en el registro I. El bloque de la red de lógica de control genera todas las funciones de control para el computador.

La configuración de la red de lógica de control en la Figura 11-6 completa el diseño del control con componentes alambrados. Esta configuración consiste de las compuertas combinacionales que generan las 24 funciones de control listadas en la Tabla 11-10. Las funciones de Boole listadas como funciones de control especifican las ecuaciones de Boole de las cuales se puede deducir el circuito combinacional. Este circuito no se dibujará aquí pero puede obtenerse de las 24 funciones de Boole que definen las variables de control R, W, a_1 *hasta* a_8, b_1, b_2, b_3, c_1, c_2, d_1, d_2, i_1, e_1, e_2, f_1, f_2, s_1 y u_1.

Control por PLA

El control por PLA es similar al método de registro de secuencia y decodificador, excepto que todos los circuitos combinacionales se configuran dentro

del PLA. Los dos decodificadores se incluyen dentro de la configuración del PLA ya que ellos son circuitos combinacionales. El número de salidas de control es 24. El número total de terminales de entrada PLA es también 24. Un PLA de 24 entradas y 24 salidas puede no estar disponible en cápsula de CI comercial. Por esta razón, la unidad de control debe distribuirse de tal manera que pueda configurarse con un número mínimo de CI PLA.

Una forma de repartir el control es de acuerdo a las tablas de función presentadas en la Sección 11-5. Las proposiciones de trasferencia entre registros de esta sección se listan en las Tablas 11-5 hasta 11-9. El control PLA repartido de acuerdo a estas tablas se muestra en la Figura 11-8. Esta configuración remplaza el control de los componentes alambrados de la Figura 11-6.

La Figura 11-8 muestra tres PLA y dos registros para la unidad de control. Los dos decodificadores no son necesarios aquí ya que se configuran dentro del PLA. Nótese que no hay conexiones de las salidas de cualquier PLA a las entradas del registro de secuencia G. Una conexión de realimentación no es necesaria porque el registro G es un contador y el siguiente estado se predetermina a partir de la secuencia de cuenta continua. El PLA 1 configura las variables de control listadas en la Tabla 11-5 (ciclo de búsqueda) y la Tabla 11-6 (operaciones comunes para el ciclo de ejecución). Estas variables de control dependen de las variables de tiempo G, el código de operación de I y el control del ciclo en F. El PLA 2 configura las funciones de control listadas en la Tabla 11-7 (ejecución de las instrucciones de referencia de memoria). Estas funciones de control tienen las mismas variables de entrada que en el PLA 1 con la adición de la variable binaria B_z. Recuérdese que B_z es una variable binaria igual a 1 cuando el registro B contiene sólo ceros.

El tercer PLA genera las funciones de control de referencia entre registros y de entrada-salida listadas en las Tablas 11-8 y 11-9. Estas funciones de control tienen dos variables comunes:

$r = q_6 t_3$ para las operaciones de referencia entre registros.

$p = q_7 t_3$ para las operaciones de entrada-salida.

Estas dos variables comunes son generadas en el PLA 1 y se aplican como entradas al PLA 3. Las otras entradas al tercer PLA provienen del registro B (bits $1 - 12$) y de otras condiciones del bit de condición.

La variable de control c_1 incrementa el contador de programa. Esta variable de control se genera en todos los tres PLA. Las tres salidas deben combinarse con una compuerta externa OR para producir una sola salida. Esta salida se aplica al terminal de entrada de incremento del PC.

La derivación de las tablas de programa para los tres PLA completa el diseño del control. La tabla de programa del PLA 1 puede obtenerse a partir de las funciones de control listadas en las Tablas 11-5 y 11-6. Estas funciones se repiten de nuevo en la Tabla 11-11 por conveniencia. Algunas de las funciones han sido simplificadas para ser agregadas en la tabla de programa. Por ejemplo la variable de control de lectura R fue listada originalmente como:

$$R = F't_1 + F(q_0 + q_1 + q_3)t_3$$

Las variables de salida decodificadas q_0, q_1 y q_3 son una función de las variables en el registro I y pueden ser simplificadas de la siguiente manera:

$$q_0 + q_1 + q_3 = I_3'I_2'I_1' + I_3'I_2'I_1 + I_3'I_2I_1 = I_3'I_1 + I_3'I_2'$$

Como el PLA acepta las variables I en vez de las variables q es más conveniente usar la función de dos términos en vez de una de tres términos. La variable de control f_1 se simplifica de una manera similar. Las otras varia-

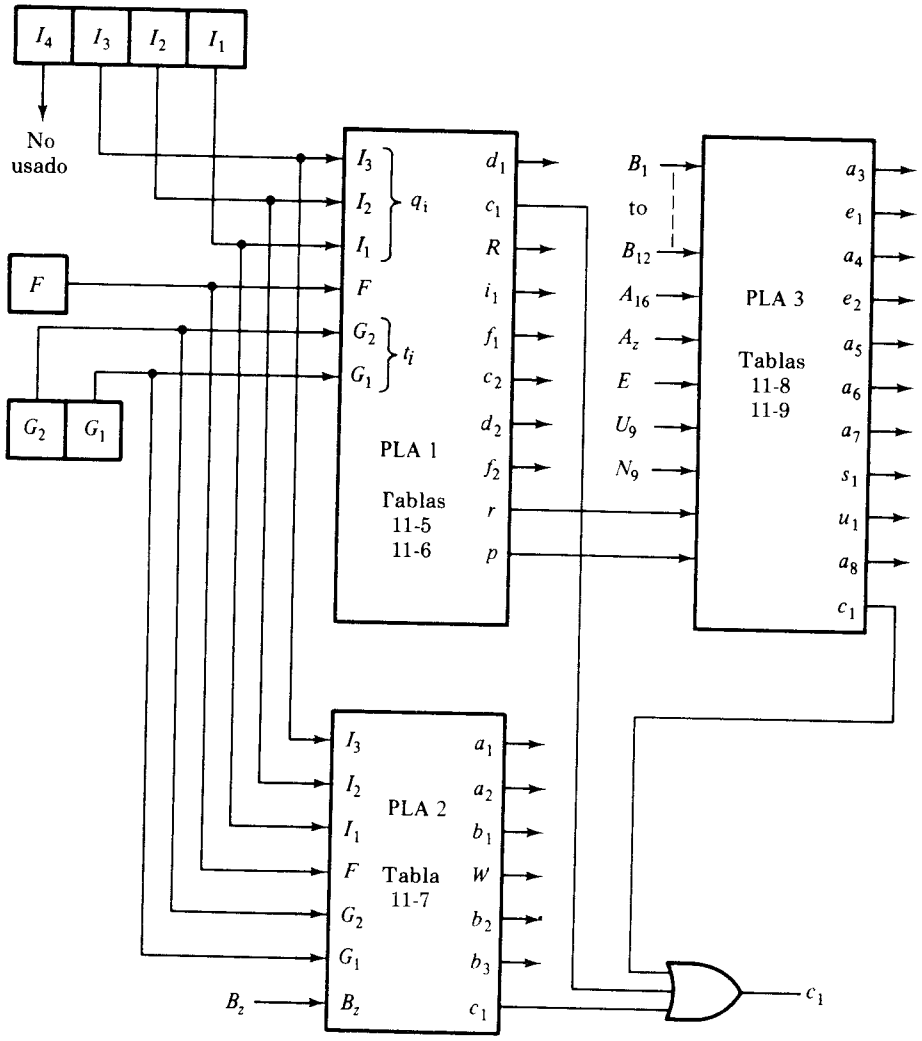


Figura 11-8 Control PLA para computador

Tabla 11-11 Funciones de control para el PLA 1

$d_1 = F't_0$:	$MAR \leftarrow PC$
$c_1 = F't_1$:	$PC \leftarrow PC + 1$
$R = F't_1 + F(I_3'I_1 + I_3'I_2')t_1$:	$B \leftarrow M$
$i_1 = F't_2$:	$I \leftarrow B(OP)$
$f_1 = F'(I_3' + I_2'I_1')t_3$:	$F \leftarrow 1$
$c_2 = q_5t_3$:	$PC \leftarrow B(AD)$
$d_2 = Ft_0$:	$MAR \leftarrow B(AD)$
$f_2 = Ft_3$:	$F \leftarrow 0$
$r = q_6t_3$:	Registro de referencia
$p = q_7t_3$:	Entrada-salida

bles de Boole necesitan una traducción de la designación t a un estado en el registro de secuencia G y de la designación q al código de operación correspondiente en el registro I.

La tabla de programa para el PLA 1 se da en la Tabla 11-12. El PLA tiene 6 entradas, 12 términos producto y 10 salidas. Las entradas para G_2 y G_1 son 00, 01, 10 y 11 y corresponden a las variables de tiempo t_0, t_1, t_2 y t_3 respectivamente. La entrada para I_3, I_2 e I_1 es un número binario igual al valor del suscrito i en q_i a no ser que se simplifique la función. Nótese que el registro I tiene cuatro bits pero I_4 no se usa ya que siempre es 0. El procedimiento para obtener una tabla de programa PLA a partir de un conjunto de funciones de Boole se explica en la Sección 5-8.

La tabla de programa para el PLA 2 puede derivarse de manera similar, pero no se incluye aquí. El tercer PLA requiere 12 términos AND y una compuerta OR de 6 entradas (para generar la variable de control c_1). Esta

Tabla 11-12 Tabla de programa para el PLA 1

Término del producto	Entradas						Salidas										
	I_3	I_2	I_1	F	G_2	G_1	d_1	c_1	R	i_1	f_1	c_2	d_2	f_2	r	p	
1	–	–	–	0	0	0	1	–	–	–	–	–	–	–	–	–	$F't_0$
2	–	–	–	0	0	1	–	1	1	–	–	–	–	–	–	–	$F't_1$
3	0	–	1	1	0	1	–	–	1	–	–	–	–	–	–	–	$FI_3'I_1t_1$
4	0	0	–	1	0	1	–	–	1	–	–	–	–	–	–	–	$FI_3'I_2't_1$
5	–	–	–	0	1	0	–	–	–	1	–	–	–	–	–	–	$F't_2$
6	0	–	–	0	1	1	–	–	–	–	1	–	–	–	–	–	$F'I_3't_3$
7	–	0	0	0	1	1	–	–	–	–	1	–	–	–	–	–	$F'I_2'I_1't_3$
8	1	0	1	–	1	1	–	–	–	–	–	1	–	–	–	–	q_5t_3
9	–	–	–	1	0	0	–	–	–	–	–	–	1	–	–	–	Ft_0
10	–	–	–	1	1	1	–	–	–	–	–	–	–	1	–	–	Ft_3
11	1	1	0	–	1	1	–	–	–	–	–	–	–	–	1	–	q_6t_3
12	1	1	1	–	1	1	–	–	–	–	–	–	–	–	–	1	q_7t_3

parte del control puede configurarse más económicamente con compuertas SSI o con un arreglo de compuertas programables a voluntad (FPGA, field-programmable gate array). El FPGA es similar al FPLA field-programmable logic array) en concepto, excepto que contiene solamente compuertas AND programables. Un FPGA típico tiene 9 compuertas AND o (NAND) que comparten 16 entradas comunes.* Se requieren dos circuitos integrados FPGA para remplazar el PLA 3 en la Figura 11-8. En la compuerta OR externa puede combinarse con otras líneas que generan la variable c_1.

Control del microprograma

La organización de la unidad de control para el computador es más adecuada para el control del PLA que para el control del microprograma, principalmente por la forma como fueron formuladas originalmente las instrucciones de referencia entre registros. La configuración del control del microprograma que se va a desarrollar aquí, configura las funciones de control para el ciclo de búsqueda y las instrucciones de referencia de memoria. Las operaciones de referencia de registro de entrada-salida pueden configurarse más eficientemente con un control de componentes interconectados o un control PLA.

El control del microprograma no necesita los registros I, G y F. El código de operación está en $B(OP)$ y al final del ciclo de búsqueda puede ser usado para especificar una dirección de macrooperación para la memoria de control sin necesidad de un registro I. Las variables de tiempo generadas en el registro de secuencia G pueden ser remplazadas por una secuencia de pulsos de reloj que leen microinstrucciones consecutivas de la memoria de control. La trasferencia del ciclo de búsqueda al ciclo de ejecución puede hacerse en la memoria de control por medio de una microinstrucción de bifurcación, la cual trasfiere el control al siguiente ciclo sin usar el flip-flop F. La configuración del control de microprograma que se va a desarrollar aquí remplaza todo el control de componentes interconectados de la Figura 11-6 (excepto por el registro B).

Repasando las Tablas 11-5, 11-6 y 11-7 se nota que todas las microinstrucciones pueden secuenciarse incrementando la dirección de la memoria de control, excepto cuando se va a ejecutar una instrucción de referencia de memoria particular o para regresar al ciclo de búsqueda. Un rutina particular de instrucción de referencia de memoria puede ser accesible con una dirección de macrooperación externa. Si se comienza el ciclo de búsqueda a partir de la dirección 0, es posible bifurcarlo al ciclo de envío borrando el registro de dirección de la memoria de control CAR. Por tanto, la parte de secuencia de dirección del control del microprograma necesita solamente tres operaciones:

1. Incrementar el CAR para leer la siguiente microinstrucción en secuencia.
2. Borrar el CAR para iniciar el ciclo de búsqueda.

*El CI tipo 82S103 de Signetics.

3. Entregar la trasformación de bits del $B(OP)$ a una dirección externa del CAR (control memory address register).

Un control de microprograma para el computador se muestra en la Figura 11-9. La memoria de control ROM tiene 32 palabras de 7 bits cada una. Los primeros cuatro bits se codifican para producir 16 combinaciones de bits, cada uno para cada función de control. Aunque el computador tiene 24 funciones de control 16 son suficientes para generar aquellas funciones de control asociadas con el ciclo de búsqueda y las instrucciones de referencia de memoria. En vez de usar 16 bits de ROM para especificar 16 salidas, se escoge usar solamente 4 bits y decodificarlos mediante un decodificador de 4 a 16 líneas para producir hasta 16 variables de salida distinguibles. Este esquema ahorra bits de ROM pero requiere de un decodificador externo. Este limita también la capacidad de las microinstrucciones porque solamente se puede especificar una función de control en cualquier microinstrucción dada.

La parte de secuenciamiento de direcciones de la unidad de microprograma no requiere un multiplexor para seleccionar las condiciones del bit de condición. Hay solamente un bit de condición que se debe considerar y se mostrará más adelante cómo puede incluirse en un circuito externo. No hay necesidad para un campo de dirección en la microinstrucción porque no se presentan necesidades de bifurcación excepto para el regreso al comienzo del ciclo de búsqueda o la trasferencia a un registro de dirección. Los últimos tres bits de la microinstrucción determinan la siguiente dirección. El bit 7 incrementa el control del registro de dirección. El bit 6 borra el CAR, lo cual causa un regreso al ciclo de búsqueda. El bit 5 carga una dirección externa al CAR. La dirección de entrada debe contener 5 bits porque el ROM tiene $32 = 2^5$ palabras. Tres de estos bits vienen de la parte del registro B que retiene el código de operación. Los últimos dos bits son siempre iguales a 11. Esta es una trasformación de código de los bits del código de operación de la instrucción a una dirección externa de la memoria

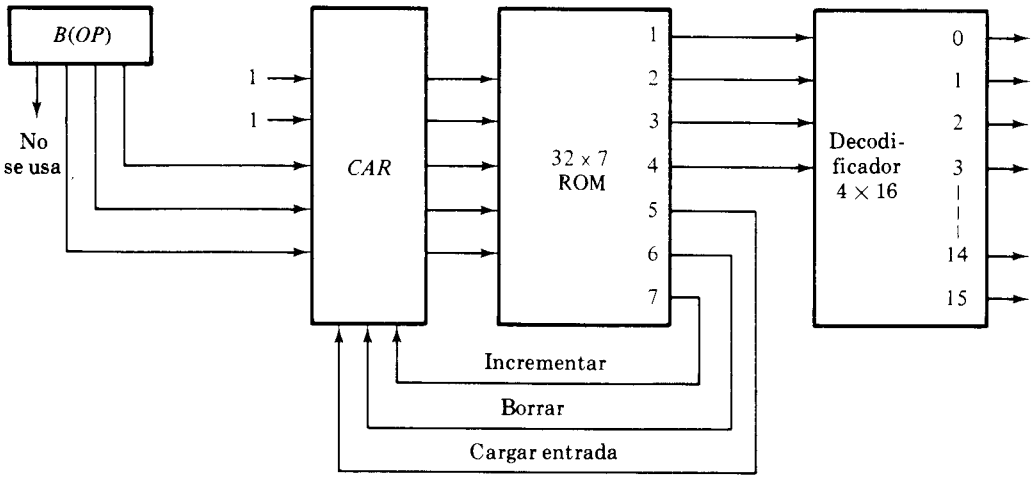


Figura 11-9 Unidad de control del microprograma para el computador

de control. Esta trasformación causa que la instrucción AND, cuyo código de operación es 000, cambie a una dirección del CAR igual a 00011. La instrucción ADD se trasforma de 001 a 00111 y así sucesivamente hasta una instrucción de entrada-salida cuyo código de operación es 111 y cuya trasformación de dirección es 11111. El bit más significativo en $B(OP)$ no se usa porque es siempre 0.

La unidad de control del microprograma, mostrada en la Figura 11-9, es muy simple y requiere solamente tres circuitos MSI. Debido a su simplicidad no es muy flexible y como se muestra subsecuentemente requiere circuitos adicionales para una configuración completa de la unidad de control.

Las microinstrucciones para el ciclo de búsqueda y de ejecución de las instrucciones de referencia de memoria se listan en las Tablas 11-5, 11-6 y 11-7. Las microoperaciones para los registros I y F no son necesarias ya que esos registros no se usan. Las microoperaciones restantes y sus funciones de control codificadas se listan en la Tabla 11-13. Los primeros cuatro bits de una palabra ROM en la memoria de control producen 16 combinaciones y cada combinación especifica una microoperación. Las combinaciones de sólo ceros y sólo unos no inicia una microoperación. Las otras 14 combinaciones se decodifican para entregar variables de control para las microoperaciones listadas. La salida del decodificador 14 inicia la operación de escritura en memoria $M \leftarrow B$ y también especifica un control condicional para incrementar el PC dependiendo de la variable Bz. La razón para repetir estas dos microoperaciones en una microinstrucción se clarificará más tarde. Nótese que la microoperación de escritura de memoria se inicia con la salida 11 del decodificador y la variable de control que incrementa el PC está disponible de la salida 2 del decodificador.

Tabla 11-13 Codificación de los bits de ROM para las microoperaciones

Bits de ROM				Salida del	Función	
1	2	3	4	decodificador	de control	Microoperación
0	0	0	0	0	—	Ninguna
0	0	0	1	1	d_1	$MAR \leftarrow PC$
0	0	1	0	2	c_1	$PC \leftarrow PC + 1$
0	0	1	1	3	R	$B \leftarrow M$
0	1	0	0	4	c_2	$PC \leftarrow B(AD)$
0	1	0	1	5	d_2	$MAR \leftarrow B(AD)$
0	1	1	0	6	r	Operación de referencia entre registros
0	1	1	1	7	p	Operación de entrada-salida
1	0	0	0	8	a_1	$A \leftarrow A \wedge B$
1	0	0	1	9	a_2	$A \leftarrow A + B, E \leftarrow$ arrastre
1	0	1	0	10	b_1	$B \leftarrow A$
1	0	1	1	11	W	$M \leftarrow B$
1	1	0	0	12	b_2	$B \leftarrow B + 1$
1	1	0	1	13	b_3	$B(AD) \leftarrow PC, B(OP) \leftarrow 0101, PC \leftarrow MAR$
1	1	1	0	14	W, c_1	$M \leftarrow B$, si $(B_z = 1)$ entonces $(PC \leftarrow PC + 1)$
1	1	1	1	15	—	Ninguna

Tabla 11-14 Tabla de verdad del ROM para el control del microprograma

Instrucción	Dirección del ROM	Salidas del ROM							Designación simbólica	
		1	2	3	4	5	6	7	Microoperaciones	Siguiente dirección
BUSQUEDA	00000	0	0	0	1	0	0	1	$MAR \leftarrow PC$	$CAR \leftarrow CAR + 1$
	00001	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	00010	0	0	1	0	1	0	0	$PC \leftarrow PC + 1$	$CAR \leftarrow 2^2B(OP) + 3$
AND	00011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	00100	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	00101	1	0	0	0	0	1	0	$A \leftarrow A \wedge B$	$CAR \leftarrow 0$
	00110	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
SUMA	00111	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	01000	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	01001	1	0	0	1	0	1	0	$A \leftarrow A + B$, $E \leftarrow$ arrastre	$CAR \leftarrow 0$
	01010	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
STO	01011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	01100	1	0	1	0	0	0	1	$B \leftarrow A$	$CAR \leftarrow CAR + 1$
	01101	1	0	1	1	0	1	0	$M \leftarrow B$	$CAR \leftarrow 0$
	01110	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
ISZ	01111	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	10000	0	0	1	1	0	0	1	$B \leftarrow M$	$CAR \leftarrow CAR + 1$
	10001	1	1	0	0	0	0	1	$B \leftarrow B + 1$	$CAR \leftarrow CAR + 1$
	10010	1	1	1	0	0	1	0	$M \leftarrow B$,si $(B_z = 1)$ entonces $(PC \leftarrow PC + 1)$	$CAR \leftarrow 0$
BSB	10011	0	1	0	1	0	0	1	$MAR \leftarrow B(AD)$	$CAR \leftarrow CAR + 1$
	10100	1	1	0	1	0	0	1	$B(AD) \leftarrow PC$, $PC \leftarrow MAR$	$CAR \leftarrow CAR + 1$
	10101	1	0	1	1	0	0	1	$M \leftarrow B$	$CAR \leftarrow CAR + 1$
	10110	0	0	1	0	0	1	0	$PC \leftarrow PC + 1$	$CAR \leftarrow 0$
BUN	10111	0	1	0	0	0	1	0	$PC \leftarrow B(AD)$	$CAR \leftarrow 0$
	11000	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
	11001	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
	11010	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
REGISTRO	11011	0	1	1	0	0	1	0	Operación de registro	$CAR \leftarrow 0$
	11100	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
	11101	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
	11110	0	0	0	0	0	1	0	Ninguna	$CAR \leftarrow 0$
I/O	11111	0	1	1	1	0	1	0	Operación de entrada-salida	$CAR \leftarrow 0$

El microprograma para la memoria de control se da en la Tabla 11-14. Este es también la tabla de verdad para programar el ROM. Hay 32 palabras de ROM; la dirección y contenido de cada palabra se especifican en la tabla. La tabla se subdivide en nueve rutinas que muestran las microinstrucciones que pertenecen al ciclo de búsqueda y las microinstrucciones para ejecutar cada una de las instrucciones de computador. La columna de la designación simbólica presenta el microprograma en forma simbólica y la secuencia de direcciones para el *CAR*.

El ciclo de búsqueda comienza con la dirección 0. Las tres microoperaciones consecutivas en la rutina de búsqueda trasfieren el contenido del *PC* al *MAR*, leen la instrucción al registro *B* e incrementan el *PC*. En la dirección 2 (0010) el bit 5 de la microinstrucción es igual a 1. El mismo pulso de reloj que incrementa el *PC* también ejecuta la microoperación:

$$CAR \leftarrow 2^2B(OP) + 3$$

$B(OP)$ contiene los tres bits del código de operación. Estos bits se desplazan doblemente a la izquierda (multiplicando por 2^2) y el binario 3 (11) se agrega para formar una dirección para el *CAR*. La dirección recibida en el *CAR* trasfiere el control a una de las rutinas listadas en la tabla y el control continúa ejecutando la instrucción específica. La configuración de esta trasformación de código se ilustra en la Figura 11-9.

Esta configuración asigna cuatro palabras de ROM para cada instrucción, excepto para la instrucción I/O. Por ejemplo, la instrucción ISZ tiene el código de operación 011. El comienzo de la rutina que ejecuta esta instrucción está en la dirección $4 \times 3 + 3 = 15$ la cual es el binario 01111. Las cuatro palabras ROM para esta rutina están en las direcciones 15, 16, 17 y 18. No se puede usar la palabra de la dirección 19 porque esta dirección contiene la primera microinstrucción para la rutina BSB. Como no hay capacidad de bifurcación en esta unidad de microprograma, no se puede bifurcar a una palabra de ROM no usada; por tanto cada rutina debe completarse con cuatro microinstrucciones o menos.

La rutina AND puede configurarse con tres microinstrucciones. La dirección de la instrucción se trasfiere al *MAR*, el operando se lee de la memoria a *B* y se ejecuta la microoperación AND entre los registros *A* y *B*. La última microinstrucción en la dirección 5 (00101) tiene el bit 6 igual a 1. Esto causa que el *CAR* se borre y el control regrese a la dirección 0 para comenzar de nuevo el ciclo de búsqueda. Las primeras dos microinstrucciones de la rutina AND tienen el bit 7 igual a 1, lo cual causa que el *CAR* se incremente. La última palabra de esta rutina en la dirección 6 no se usa. Esta palabra no puede dejarse vacía ya que se debe especificar algo para la tabla de verdad del ROM. La mejor manera de ocupar esta palabra es no especificar microoperaciones en los bits 1 hasta 4 y borrar el *CAR* con el bit 6. De esta manera, si ocurre una falta y la memoria de control se encuentra en la dirección 6 no se ejecutará ninguna operación y el control regresará al ciclo de búsqueda.

Las rutinas de ADD y STO necesitan tres microinstrucciones. La instrucción BSB usa todas las cuatro palabras disponibles en la rutina. La instrucción BUN necesita solamente una microinstrucción. Una instrucción

de referencia de registro inicia una variable de control r, la cual puede ser usada conjuntamente con un bit en el registro B para iniciar una de las operaciones especificadas. Lo mismo se aplica a la instrucción (I/O) de entrada-salida.

La rutina ISZ necesita cuatro microoperaciones y una operación condicional dependiente del valor de B_z. Esta impone un problema ya que solamente hay cuatro palabras de ROM disponibles para esta rutina y la configuración del microprograma no tiene facilidad para comprobar el estado del bit de condición. Este problema puede resolverse incluyendo dos microoperaciones y una microinstrucción y comprobando el bit de condición con una compuerta AND externa. Para compensar esta configuración no ortodoxa se agrega un circuito externo como se ilustra en la Figura 11-10. El decodificador ROM tiene dos terminales de salidas para la operación de escritura de memoria $M \leftarrow B$: uno en la salida 11 y el otro en la salida 14. Esas dos salidas se aplican a una compuerta OR externa para entregar una salida común. El terminal de salida 14 del decodificador se habilita durante la cuarta microinstrucción de la rutina ISZ. Esta salida se aplica a una compuerta AND externa con el bit de condición B_z para producir la función de control de incremento del PC. La salida 2 del decodificador especifica también un incremento del PC. Algunas de las operaciones en las instrucciones de referencia entre registro y de entrada-salida especifican igualmente esta operación. Las tres salidas deben aplicarse a una compuerta OR para conformar una sola salida para incrementar el PC. Las variables r y p del decodificador del ROM se usan conjuntamente con otras condiciones del bit de condición para generar las variables de control restantes para el computador. Esas variables de control pueden ser generadas con una configuración externa de componentes o con un PLA tal como se indica en el diagrama.

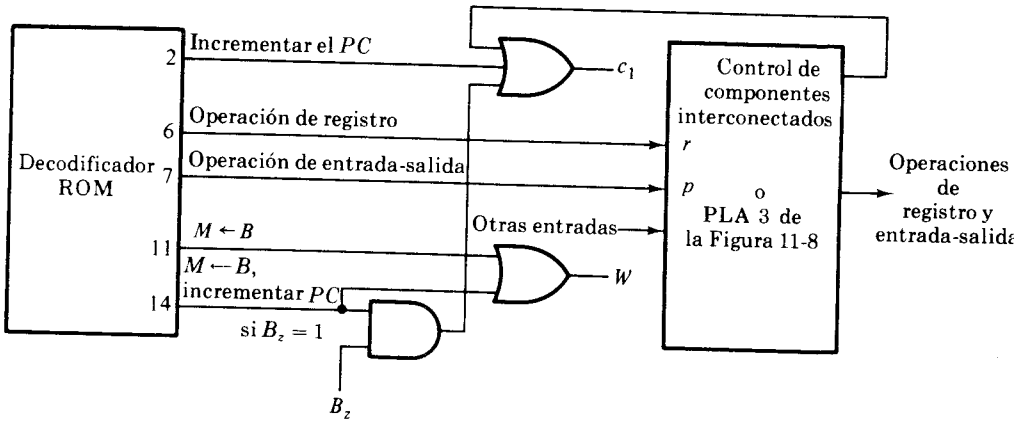


Figura 11-10 Circuitos adicionales para un control de microprograma

11-8 CONSOLA DEL COMPUTADOR

Cualquier computador tiene un panel de control o consola con interruptores y lámparas para permitir comunicación manual y visual entre el operador

y el computador. Esta comunicación es necesaria para comenzar la operación del computador (bootstrapping) y para propósitos de mantenimiento. Para completar se enumera un conjunto de funciones de la consola útiles para el computador aunque no se muestren los circuitos necesarios para configurar estas funciones.

Las lámparas indican al operador la condición de los registros del computador. La salida normal de un flip-flop conectado a una lámpara indicadora causará que la lámpara alumbre cuando el flip-flop se pone a 1 y se apague cuando el flip-flop se borra. Los registros cuyas salidas van a ser observadas en la consola del computador son: A, B, PC, MAR, I, E, F y S. Cuando se muestra el número total del flip-flop se encuentra que es necesario 63 lámparas indicadoras.

Un conjunto de interruptores y sus funciones para la consola pueden incluir lo siguiente:

1. Dieciséis interruptores de "palabra" para establecer manualmente los bits de una palabra.
2. Un interruptor de "comienzo" para preparar el flip-flop S. La señal de este interruptor borra el flip-flop F, N_9, U_9 y el registro G.
3. El interruptor de "parada" para borrar el flip-flop S. Para asegurar que se complete esta instrucción la señal que viene del interruptor se aplica conjuntamente con la función de Boole $(F + q_5 + q_6 + q_2 + q_7)t_3$ a una compuerta AND antes de que se aplique para el borrado de la compuerta S.
4. Un interruptor de "cargar dirección" para trasferir una dirección al registro PC. Cuando se activa este interruptor, el contenido de los 12 interruptores de "palabra" se trasfieren al PC.
5. Un interruptor de "depósito" para almacenar manualmente palabras en la memoria. Cuando se activa este interruptor, el contenido del PC se trasfiere al MAR y se inicia el ciclo de memoria. Después de 1 μs, el contenido de los 16 interruptores de "palabra" se trafieren al registro B y se incrementa el PC en 1.
6. Un interruptor de "exposición" para examinar el contenido de la palabra en la memoria. Cuando se activa este interruptor, el contenido del PC se trasfiere al MAR, se inicia un ciclo de memoria y se incrementa el PC en 1. El contenido de la palabra de memoria especificado por la dirección en el PC, está en el registro B y puede verse en las correspondientes lámparas indicadoras.

Para asegurarse que el computador no esté funcionando cuando la energía se aplica, el flip-flop S debe tener un círculo especial que lo force a una posición de borrado inmediatamente después de aplicar energía a la máquina.

REFERENCIAS

1. Mano, M. M., *Computer System Architecture*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1976.

2. *Small Computer Handbook*. Maynard, Mass.: Digital Equipment Corp., 1973.

3. Booth, T. H., *Digital Networks and Computer Systems*. Nueva York: John Wiley & Sons, Inc., 1971.

4. Hill, F. J. y G. R. Peterson, *Digital Systems: Hardware Organization and Design*. Nueva York: John Wiley & Sons, Inc., 1973.

5. Bell, C. G., J. Grason y A. Newell, *Designing Computers and Digital Systems*. Maynard, Mass.: Digital Press, 1972.

6. Kline, R. M., *Digital Computer Design*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1977.

7. Soucek, B., *Minicomputers in Data Processing and Simulation*. Nueva York: John Willey & Sons, Inc., 1972.

PROBLEMAS

11-1. Repase el conjunto de instrucciones del computador diseñado en este capítulo (Tablas 11-2, 11-3 y 11-4) y haga la lista de las instrucciones útiles para:

(a) trasferencias entre la memoria y el acumulador;

(b) trasferencias entre la entrada-salida y el acumulador;

(c) manipulaciones aritméticas;

(d) operaciones lógicas;

(e) operaciones de desplazamiento;

(f) decisiones de control basadas en condiciones de estado;

(g) subrutina de bifurcación y regreso.

11-2. Dé una lista de instrucciones para que el computador ponga a uno el flip-flop E.

11-3. (a) Haga una lista de la secuencia de instrucciones para que el computador ejecute un desplazamiento aritmético a la derecha de un número almacenado en el acumulador. El número está en la representación de signo complemento de 2. (b) Repita lo anterior para un desplazamiento aritmético a la izquierda. Indique cómo se detecta una sobrecapacidad.

11-4. Muestre que la lista de instrucciones obtenidas en el Problema 11-1(d) constituye un conjunto suficiente para configurar todas las 16 operaciones lógicas listadas en la Tabla 2-6.

11-5. (a) Escriba una secuencia de tres instrucciones que se almacenen en los lugares de memoria 1, 2 y 3. Ellas deben constatar si hay un caracter en un dispositivo de entrada y si es así, trasferirlo al acumulador. (b) Escriba una secuencia de tres instrucciones que se almacenen en los lugares de memoria 5, 6 y 7. Ellos deben constatar si el dispositivo de salida está desocupado y si es así, trasferir un caracter del acumulador.

11-6. El computador descrito en este capítulo no tiene una indicación de sobrecapacidad después de sumar dos números con signo. Asuma que los dos números agregados con la instrucción ADD están en la representación de signo complemento de 2. Describa un algoritmo en forma de flujograma para un programa de computador que sume dos números y detecte una sobrecapacidad.

11-7. El programa siguiente es una lista de instrucciones en código hexadecimal. El computador ejecuta las instrucciones comenzando por la posición hexadecimal 100.

Símbolo	Código hexadecimal	Descripción	Función
ORA	8 *m*	OR con *A*	$A \leftarrow A \vee M$
XRA	9 *m*	OR-exclusiva con *A*	$A \leftarrow A \oplus M$
SWP	A *m*	Intercambiar *A* con la memoria	$A \leftarrow M, M \leftarrow A$
SUB	B *m*	Restar *A* de la memoria	$A \leftarrow M - A$
BSA	C *m*	Bifurcar y conservar la dirección en *A*	$A \leftarrow PC, PC \leftarrow m$
BPA	D *m*	Bifurcar con *A* positivo	Si $(A \geqslant 0)$ entonces $(PC \leftarrow m)$
BNA	E *m*	Bifurcar con *A* negativo	Si $(A < 0)$ entonces $(PC \leftarrow m)$
BZA	F *m*	Bifurcar si *A* es cero	Si $(A = 0)$ entonces $(PC \leftarrow m)$

11-13. El computador diseñado en este capítulo usa un flip-flop *F* para distinguir entre los ciclos de búsqueda y ejecución. Este flip-flop no se necesita si el registro de secuencia *G* es un contador de 3 bits y su decodificador entrega ocho señales de tiempo, t_0 hasta t_7. El registro *G* puede borrarse tan pronto como se complete la ejecución de la instrucción. (Esta es la forma como fue diseñado el control en el computador sencillo de la Sección 8-12.)

(a) Revise las Tablas 11-5, 11-6 y 11-7 para estar de acuerdo con este nuevo esquema de control.

(b) Determine el tiempo de ejecución de cada instrucción incluyendo el tiempo de búsqueda de la instrucción.

11-14. Haga una lista de las proposiciones de trasferencia entre registros para la ejecución de las instrucciones que se listan a continuación. Asuma que el computador no tiene un flip-flop *F*, pero que el registro de secuencia *G* tiene 16 variables de tiempo t_0 hasta t_{15}. El registro *G* debe borrarse cuando se complete la ejecución de la instrucción. El ciclo de búsqueda para el computador es ahora:

$$t_0: \quad PC \leftarrow MAR$$

$$t_1: \quad B \leftarrow M, PC \leftarrow PC + 1$$

$$t_2: \quad I \leftarrow B(OP)$$

Cada una de las siguientes instrucciones comienzan el ciclo de ejecución a partir de la variable de tiempo t_3. La última proposición incluye la microoperación $G \leftarrow 0$.

Símbolo	Código hexadecimal	Descripción	Función
SBA	8 *m*	Substraer de *A*	$A \leftarrow A - M$
ADM	9 *m*	Agregar a la memoria	$M \leftarrow A + M$ (*A* no cambia)
BEA	A *m*	Bifurcar si *A* es igual	Si $(A = M)$ entonces $(PC \leftarrow m)$ (*A* no cambia)

11-15. Compare las proposiciones de trasferencia entre registros del registro A listadas en la Tabla 11-10 con el acumulador diseñado en la Sección 9-10. Diseñe una etapa típica del registro A para el computador usando el procedimiento esbozado en la Sección 9-10. Incluya el circuito para la variable A_z de detección de cero.

11-16. Dibuje las compuertas lógicas que generan las funciones de control a_1 hasta a_8 para el registro A (Tabla 11-10).

11-17. Comenzando por la proposición de trasferencia entre registros dada en la Tabla 11-10 para el flip-flop E derive las funciones de entrada de Boole para E. Use un flip-flop JK.

11-18. Una manera de simplificar un circuito cuando se usa el método de trasferencia entre registros es usar los caminos comunes mientras se desarrolla la lista de proposiciones. Para ilustrar con un ejemplo particular considere el multiplexor para la entrada del PC en la Figura 11-7. Este multiplexor no sería necesario si se puede remplazar la proposición:

$$Fq_4t_2: \quad PC \leftarrow MAR$$

por la proposición:

$$Fq_4t_2: \quad PC \leftarrow B(AD)$$

en la instrucción BSB de la Tabla 11-7. Explique por qué puede hacerse esto y cómo resulta en la eliminación del multiplexor del diagrama de bloque del computador.

11-19. Un contador de 4 bits con carga en paralelo se encapsula en un circuito integrado. ¿Cuántos CI se necesitan para construir los siguientes registros de computador: PC, MAR, I y G?

11-20. Diseñe el registro G del computador usando un contador de 4 bits con carga en paralelo del tipo mostrado en la Figura 7-19.

11-21. Dé una lista de la tabla del programa del PLA 2 de la Figura 11-8.

11-22. Cambie la instrucción AND del computador a una instrucción OR y modifique el microprograma de la Tabla 11-14 para que esté acorde. Asigne la microoperación OR a la salida 15 del decodificador en la Tabla 11-13.

11-23. Cambie la instrucción BSB del computador a la instrucción BSA definida en el Problema 11-12. Modifique el microprograma de la Tabla 11-14 para estar acorde con ese cambio. La codificación de los bits de ROM en la Tabla 11-13 pueden necesitar un cambio también.

11-24. Diseñe una unidad de control de microprograma para un computador que configure el ciclo de búsqueda y ejecución de las instrucciones de referencia de memoria listadas en la Tabla 11-2 y el Problema 11-12. Incluya dos salidas para la referencia de registros y las operaciones de entrada-salida.

Diseño del sistema del microcomputador

12

12-1 INTRODUCCION

Un sistema digital se define por los registros que contiene y las operaciones que hace con la información binaria almacenada en ellos. Una vez que se ha especificado un sistema digital, el papel del diseñador es desarrollar los materiales que configuran la secuencia requerida de operaciones. El número de microoperaciones diferentes de un sistema dado es finito. La complejidad para el diseño es una secuencia de operaciones para lograr la tarea necesaria de procesamiento de datos. Esta abarca la formulación de las funciones de control o el desarrollo del microprograma. Una tercera alternativa es usar un microcomputador para configurar el sistema digital. Con un microcomputador, la secuencia de operaciones puede formularse con un conjunto de instrucciones que constituyen un programa.

Un sistema digital puede ser construido por medio de los circuitos MSI tales como registros, decodificadores, ALU, memoria y multiplexores. Tal sistema hecho a la medida tiene la ventaja que se ajusta a las necesidades de una aplicación particular. Sin embargo, un sistema digital construido con circuitos MSI podría requerir un gran número de circuitos integrados. Sin embargo, cualquier modificación que pueda necesitarse, una vez haya sido construido el sistema, se debe lograr por medio de cambios de alambrado entre los componentes.

Algunos sistemas digitales son adecuados para el diseño del LSI con componentes tales como la unidad de proceso, el secuenciador del microprograma y la unidad de memoria. Estos sistemas pueden ser microprogramados para adecuarse a especificaciones requeridas. El método del microprograma opera a nivel de trasferencia entre registros y debe especificar cada microoperación en el sistema. La organización del LSI microprogramado usa menos CI que la configuración con MSI.

El número de CI puede reducirse aún más si el sistema digital es adecuado para ser construido con componentes LSI del microcomputador. Estos componentes pueden ser clasificados por funciones de la siguiente manera:

1. Un *microprocesador*, el cual es una unidad central de proceso (CPU) encapsulado en una pastilla LSI.

2. La *memoria de acceso aleatorio* (RAM) y la *memoria de sólo lectura* (ROM) o circuitos integrados que pueden combinarse para formar cualquier tamaño de memoria necesaria para una aplicación.
3. Las unidades *programables de interconexión* cuya función es interconectar entre el CPU o la memoria una amplia variedad de dispositivos de entrada y salida.

El usuario puede interconectar esos componentes LSI para formar un sistema de microcomputador que se ajuste a las necesidades del diseño y que reduzca drásticamente el número de CI.

Un microprocesador combinado con los módulos de memoria y de interconexión se llama *microcomputador*. La palabra *micro* se usa para indicar el pequeño tamaño físico de los componentes integrantes. La segunda parte de la palabra en micro*procesador* y micro*computador* es lo que realmente los diferencia. *Procesador* se usa para indicar aquella sección del sistema, la cual ejecuta las funciones básicas para realizar instrucciones y procesar datos de la manera especificada por el programa. Esta parte se llama usualmente el CPU. El término *microcomputador* se usa para indicar un sistema de computador de pequeño tamaño consistente de tres unidades básicas: CPU, memoria e interconexión de entrada-salida. El microprocesador se encapsula comúnmente en un CI y se llama *pastilla de microprocesador*. Un microcomputador se refiere, en la mayoría de los casos, a una interconexión con componentes LSI. Por otra parte, algunas pastillas microprocesadoras incluyen dentro de la cápsula no solamente el CPU sino una parte de la memoria. Tal componente LSI se llama algunas veces *microcomputador de una cápsula*.

Un microcomputador puede ser usado como un computador para propósitos generales de bajo costo, para proporcionar capacidades similares a aquellas de cualquier otro sistema de computador. Aunque ésta es una aplicación importante, no es la que se quiere enfatizar. En muchas aplicaciones, el microcomputador se usa como un sistema general para propósitos especiales para proporcionar las operaciones de trasferencia entre registros del sistema. Este tiene la ventaja que pocas cápsulas LSI remplazan una gran cantidad de circuitos MSI que serían necesarias para generar estas operaciones. Otra ventaja es que las operaciones de trasferencia entre registros para el sistema pueden especificarse con un programa. El programa de una aplicación para propósitos especiales es inalterable y por esta razón puede ser almacenado en una memoria de sólo lectura. Una vez que un programa fijo reside en una ROM no hay diferencia de comportamiento entre un sistema digital y el diseño con base en los materiales para un cliente.

La característica más importante del microcomputador es que un sistema digital para propósito especial, de aplicación única, puede ser diseñado para escribir un programa para un computador digital para propósitos generales. La ejecución de los programas fijos e inalterables causa que el microcomputador se comporte de una manera preestablecida, justamente como se comportaría un sistema digital correspondiente basado en MSI. Este método de diseño digital no era económicamente factible de configurar, antes del desarrollo de los componentes de microcomputador pequeños y de bajo costo.

El programa almacenado en la parte de la ROM de un sistema de microcomputador es un programa de computador que no necesita alteraciones. Como la RAM es una memoria volátil, al cortar el suministro de potencia y activarlo de nuevo, se destruye la información binaria almacenada en ella. La ROM es una memoria no volátil y el programa almacenado en ella está disponible cada vez que se le suministre potencia. Por esta razón, la parte de ROM de un sistema de microcomputador se llama también la memoria de *programa*.

En este punto se debe distinguir entre un microprograma y un microcomputador. Aunque ambos usen la palabra *micro*, el primero se deriva del concepto de las microoperaciones mientras que el segundo se refiere al tamaño pequeño de los componentes. Ambos usan una ROM para almacenar un programa que especifica las operaciones en el sistema. Un microprograma almacenado en la memoria de control configura la unidad de control en el CPU. Las instrucciones almacenadas en un microcomputador pueden ser consideradas como macrooperaciones para el CPU en vez de microinstrucciones para los registros de proceso. Además, el microprograma con palabras se refiere a la forma como se ha configurado la unidad de control. Un microcomputador es un computador de tamaño pequeño cuyo CPU puede o no tener una unidad de control de microprograma.

El microcomputador de bajo costo y pequeño tamaño ha cambiado la dirección del diseño lógico digital. En vez de realizar un grupo de operaciones de trasferencia entre registros por funciones de control o un microprograma, se realizan funciones lógicas especificando un conjunto de instrucciones, las cuales se almacenan en una ROM y se ejecutan en el CPU microprocesador. Este método de diseño puede ser clasificado como un método *lógico programable* ya que las operaciones secuenciales son especificadas con un programa almacenado en la memoria.

El microprocesador es un componente central en un sistema de microcomputador. La cantidad y tipo de memoria en el sistema, lo mismo que la naturaleza de las unidades de interfase de I/O que se usan son una función de la aplicación particular. El programa fijo que reside en la ROM de un sistema de microcomputador particular es también dependiente de una aplicación específica.

El diseño de un sistema de microcomputador puede ser dividido en dos partes: diseño de materiales y diseño de programación. El diseño de los materiales consiste en la conexión de los componentes físicos para producir un sistema digital completo. El diseño de la programación trata del desarrollo de los programas para una aplicación particular. Escribir programas para un microcomputador es esencialmente igual que hacerlo para cualquier otro computador. La única diferencia es que el programador de microcomputador debe estar familiarizado con la configuración de los materiales y debe tener en cuenta los problemas asociados con la aplicación particular. Escribir programas para un computador establecido para propósitos generales incluye por lo general procedimientos de cómputo que requieren muy pocos conocimientos de la construcción de los materiales del computador en sí, en caso de que los haya.

Este capítulo abarca los aspectos de los materiales de los microcomputadores sin tener en consideración los problemas de programación. Escribir

programas para un microcomputador es similar a escribir un microprograma para la memoria de control, excepto que se debe usar el conjunto de instrucciones para el microprocesador comercial seleccionado. El estudio del diseño de los materiales es un tema que por sí solo podría llenar todo un volumen.

En este capítulo se definen primero varios componentes de un sistema de microcomputador y la forma como se comunican entre sí. La organización del microprocesador típico se presenta entonces y sus operaciones internas y externas se explican. Algunas características importantes comunes a todos los microprocesadores son discutidas. Se muestra entonces la organización de la sección de memoria y se explican varios tipos de unidades de interconexión usados comúnmente en el diseño de sistemas de microcomputadores.

12-2 ORGANIZACION DEL MICROCOMPUTADOR

Un sistema típico de microcomputador consiste de un microprocesador más memoria y una interconexión I/O. Los diferentes componentes que forman el sistema están enlazados por medio de buses que trasfieren instrucciones, datos, direcciones e información de control entre los componentes de CI. La Figura 12-1 muestra el diagrama de bloque de un sistema de microcomputador. Típicamente el microcomputador tiene un solo microprocesador. Si

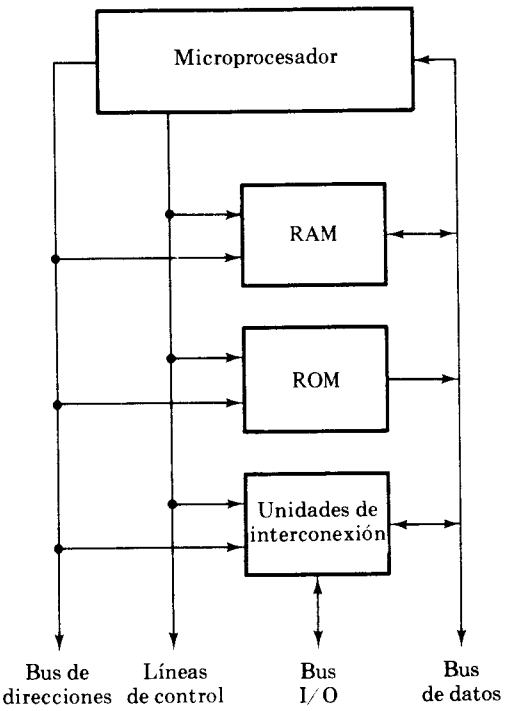


Figura 12-1 Diagrama de bloque del sistema de microcomputador

se incluyen varios procesadores, se tiene entonces un sistema multiprocesador, el cual es una posibilidad válida. Un número de cápsulas RAM y ROM son combinadas para formar un tamaño dado de memoria. Las unidades de interconexión se comunican con dispositivos externos a través del bus I/O. En un tiempo dado el microprocesador selecciona una de las unidades por medio del bus de direcciones. Los datos se trasfieren de la unidad seleccionada al microprocesador vía el bus de datos. La información de control se trasfiere usualmente por medio de líneas individuales, cada una especificando una función de control particular.

El propósito del microprocesador es suministrar un CPU que interprete códigos de instrucción recibidos de la memoria y ejecutar operaciones aritméticas, lógicas y de control basadas en datos almacenados en registros internos, palabras de memoria o unidades de interconexión. El microprocesador contiene un número de registros, una unidad lógica aritmética, una unidad de tiempo y una lógica de control. Externamente, éste entrega un sistema de buses para trasferir instrucciones, datos e información de control hacia los módulos conectados con él. Las operaciones internas de un microprocesador típico y las funciones de las líneas de control se describen en la Sección 12-3.

La memoria de acceso aleatorio es una memoria del tipo de lectura-escritura y consiste de varios CI conectados entre sí. La RAM se usa para almacenar datos, parámetros variables y resultados intermedios que necesitan renovación y que están sujetos a cambio. La ROM consiste de un número de CI y se usa para almacenar programas y tablas constantes que no están sujetas a cambio una vez que se haya terminado la producción del sistema del microcomputador. El método de conectar pastillas de memoria al microprocesador se describe en la Sección 12-6.

Las unidades de interconexión presentan los caminos necesarios para trasferir información entre el microprocesador y los dispositivos externos de entrada y salida conectados al bus I/O. El microprocesador recibe información de condiciones y datos de los dispositivos externos por medio de la interconexión. Este responde enviando información de control y datos para los dispositivos externos por medio de la interconexión. Esta comunicación se especifica por medio de instrucciones programadas que dirigen los datos por medio del bus en el sistema del microcomputador. Los diferentes módulos de interconexión disponibles en los microcomputadores y su operación se presentan en la Sección 12-7.

La comunicación entre las compuertas LSI en el microprocesador se lleva a cabo vía el bus de direcciones y el de datos. El bus de direcciones es unidireccional desde el microprocesador a otras unidades. La información binaria que el microprocesador coloca en el bus de direcciones especifica una palabra de memoria particular en la RAM o ROM. El bus de direcciones se usa para seleccionar una de las diferentes unidades interconectadas al sistema o a un registro particular de una unidad de interconexión. Una palabra de memoria y un registro de interconexión pueden distinguirse asignando una dirección diferente a cada uno. De manera alterna, una señal de control puede usarse para especificar si la dirección en el bus es para una palabra de memoria o para un registro de interconexión. El número de líneas disponible en el bus de direcciones determina el tamaño máximo de memo-

ria que puede ser acomodado en el sistema. Para n líneas, el bus de direcciones puede especificar hasta 2^n palabras de memoria. La capacidad típica de un bus de direcciones de un microprocesador es 16, para tener una capacidad máxima de memoria de $2^{16} = 65.536$ palabras. La cantidad de memoria empleada en un sistema de microcomputador depende de la aplicación particular y a menudo es menor que la máxima disponible en el bus de direcciones.

El bus de datos trasfiere los datos del microprocesador a la memoria o interconexión y viceversa, la cual es seleccionada por el bus de direcciones. El bus de datos es bidireccional, lo cual significa que la información binaria puede fluir en cualquier dirección. Un bus de datos bidireccional se usa para ahorrar patillas en un circuito integrado. Si la unidad no usara un bus bidireccional sería necesario colocarle terminales de entrada y salida, separados en la cápsula del circuito integrado. El número de líneas en el bus de datos del microprocesador varía entre 4 y 16 siendo 8 líneas el más común.

Un grupo separado de buses de datos y de direcciones es la vía de trasferencia común encontrada en los microprocesadores. La ventaja de este esquema es que el microprocesador puede seleccionar una palabra en la memoria y trasferir la palabra de datos al mismo tiempo. Algunos microprocesadores usan un bus común el cual es multiplexado en tiempo, para trasferir direcciones o datos. Por ejemplo, una barra de 16 líneas comunes puede usarse para trasferir una dirección de 16 bits seguida de una palabra de datos de 16 bits que va a ser escrita en la memoria. La ventaja de este esquema es que se necesitan menos patillas y aún así los datos pueden ser de 16 bits de longitud. La desventaja estriba en el tiempo perdido en el uso secuencial del bus común y la necesidad de un retenedor externo para retener la dirección de la memoria. Algunos microprocesadores comparten solamente parte del bus del sistema entre datos y direcciones. Un bus de 16 líneas puede usar 8 líneas bidireccionales para trasferencia de datos y 16 líneas para trasferencia de direcciones. Esto requiere compartir el bus de datos ya que la dirección se divide entre las 8 líneas del bus de datos y las 8 líneas restantes disponibles.

En vez de usar la cápsula microprocesadora como la que se muestra en la Figura 12-1, en algunas aplicaciones se remplaza este bloque con una cápsula microcomputadora. Típicamente una cápsula microcomputadora contiene un CPU con 64 palabras de RAM y 1.024 palabras de ROM, todo encapsulado en el circuito integrado. Tiene además algunas características de interconexión. Si el sistema digital que se va a diseñar no requiere más memoria o características de interconexión adicionales, entonces el sistema microcomputador puede construirse con una sola cápsula componente microcomputadora. De esta manera, esta cápsula puede usarse como un componente de bajo costo y escaso tamaño para una aplicación independiente. La mayoría de las cápsulas microcomputadoras pueden ser expandidas con ROM externa, RAM y características de interconexión para producir una aplicación de control más poderosa. En las discusiones subsiguientes, la memoria y la interconexión estarán separados del CPU, pero debe tenerse en cuenta que algo de memoria e interconexión puede ser incluido dentro de la cápsula de CI que contiene el CPU.

Para facilitar el desarrollo de los sistemas digitales para propósitos especiales por medio de un microcomputador, muchas fuentes ofrecen una unidad microprocesadora completa en un solo tablero impreso. El microprocesador, un grupo de CI ROM, RAM y de interconexión conjuntamente con otras cápsulas MSI y SSI, necesarias para la construcción de la unidad de microcomputador, se montan en un solo tablero impreso. Los terminales de los CI se conectan por medio de alambres impresos para formar una unidad microcomputadora completa. Al usuario se le da acceso a la interconexión de los dispositivos I/O por medio de los contactos del conector del tablero. El conector tiene otros contactos para acomodar todos los buses de información y permitir expansión externa de memoria y de interconexión al tablero. La expansión de memoria e interconexión se encuentran disponibles en tableros impresos ya fabricados.

Separador del bus

El sistema de buses de un microprocesador se configura comúnmente por medio de separadores del bus (bus buffer) construidos con compuertas de tres estados. Una compuerta de tres estados es un circuito digital que tiene tres condiciones a la salida. Dos de las entradas son señales equivalentes al binario 1 ó 0, como en las compuertas convencionales. El tercer estado se llama el *estado de alta impedancia.* Este último se comporta como si la salida estuviera inhabilitada o "flotara", lo cual significa que no puede afectar ni ser afectado por alguna señal externa en el terminal. El circuito electrónico de una compuerta de tres estados se explica en la Sección 13-5 conjuntamente con la Figura 13-16.

El símbolo gráfico de una compuerta separadora de tres estados se muestra en la Figura 12-2. Esta tiene una entrada normal y una entrada de control que determina el estado de la salida. Cuando la entrada de control es igual al binario 1, la compuerta se comporta como un separador convencional con la salida igual a la entrada normal. Cuando el terminal de control es 0 se inhabilita la salida y la compuerta pasa al estado de alta impedancia, sin tener en cuenta el valor de la entrada normal. El estado de alta impedancia le imprime a la compuerta de tres estados una característica no disponible en otras compuertas. Debido a esta característica se puede conectar un gran número de compuertas de tres estados a alambres para formar un bus común de líneas sin causar efectos de carga. Sin embargo, en un tiempo dado no puede estar más de una compuerta en estado activo. Las compuertas conectadas deben ser controladas de manera que sola-

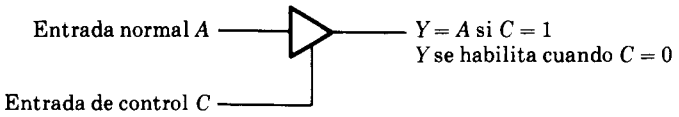


Figura 12-2 Símbolo gráfico para una compuerta separadora de tres estados

mente una compuerta de tres estados tenga acceso al bus de líneas mientras que las otras compuertas estén en el estado de alta impedancia.

Un bus bidireccional puede construirse con separadores de bus para controlar la dirección del flujo de información. Una línea del bus bidireccional se muestra en la Figura 12-3. El bus de control tiene dos líneas de selección, s_i para la trasferencia de entrada y s_o para la trasferencia de salida. Esas líneas de selección controlan los dos separadores de tres estados. Cuando $s_i = 1$ y $s_o = 0$ el separador inferior se habilita y el superior se inhabilita al pasar a un estado de alta impedancia. Esto forma una vía para los datos de entrada provenientes del bus para pasar por el separador superior y pasar luego al sistema. Cuando $s_c = 1$ y $s_i = 0$, el separador superior se habilita y el inferior pasa a un estado de alta impedancia. Esto forma una vía para los datos de salida provenientes del sistema y que pasan por la compuerta superior hacia el bus de líneas. La línea de bus puede inhabilitarse haciendo s_i y s_o igual a cero ambos, lo cual coloca a ambos separadores en estado de alta impedancia para prevenir cualquier trasferencia de información de salida o entrada a través del bus de líneas. Esta condición debe existir cuando una fuente externa esté usando el bus común para comunicarse con algún otro componente. Las dos líneas de selección pueden ser usadas para informar a los módulos externos conectados al bus del estado en el cual está el bus bidireccional en un momento dado de tiempo.

En la mayoría de los casos la capacidad de carga de un bus de microprocesador es limitada, es decir, puede soportar un número reducido de cargas externas. Cuando el bus se conecta a un gran número de unidades externas, la capacidad de carga del microprocesador debe ser reforzada con separadores externos del bus los cuales se pueden encontrar en la forma de CI. Además, cualquier componente que tiene terminales de entrada y salida separados, debe estar conectado al sistema de bus del microcomputador por medio de separadores de bus externos para poder aislar el componente cuando no se está comunicando con el bus. Así, un sistema de microcomputador necesita muy a menudo separadores de bus externos entre el microprocesador y otros componentes LSI y entre ciertos componentes LSI y el sistema de bus común.

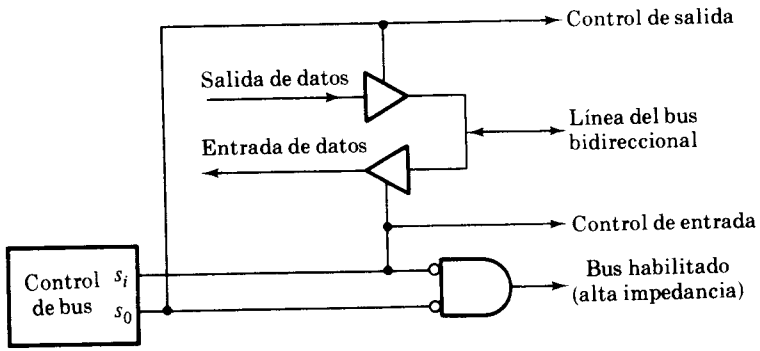


Figura 12-3 Separador de bus bidireccional

12-3 ORGANIZACION DEL MICROPROCESADOR

Para garantizar un amplio rango de aceptabilidad, un microprocesador debe tener una organización interna adecuada para una gama amplia de aplicaciones. Las organizaciones de los microprocesadores comerciales difieren entre sí, pero ellas tienen la propiedad común de una unidad procesadora central. Como tal, ellas son capaces de interpretar códigos de instrucción recibidos de la memoria y de realizar tareas de procesamiento de datos especificados por un programa. Ellas responden a los comandos de control externos y generan señales de control para ser usadas por módulos externos.

Conjunto típico de señales de control

La operación adecuada de un microprocesador requiere que se presenten ciertas señales de control y tiempo para lograr funciones específicas y que otras señales de control sean medidas para determinar el estado del microprocesador. Un conjunto típico de líneas de control disponibles en la mayoría de los microprocesadores se muestra en la Figura 12-4. Para completar el diagrama se muestra también el bus de datos, el bus de direcciones y el terminal de entrada de la fuente de poder a la unidad. Las necesidades de potencia de un microprocesador particular se especifican por el nivel de voltaje y consumo de poder que debe suministrarse para operar el CI.

El terminal de entrada del *reloj* es usado por el microcomputador para generar pulsos de reloj de multifase y producir secuencias de tiempo y control para las funciones internas. Algunos microprocesadores requieren un generador externo de pulsos de reloj para producir los pulsos. En este caso

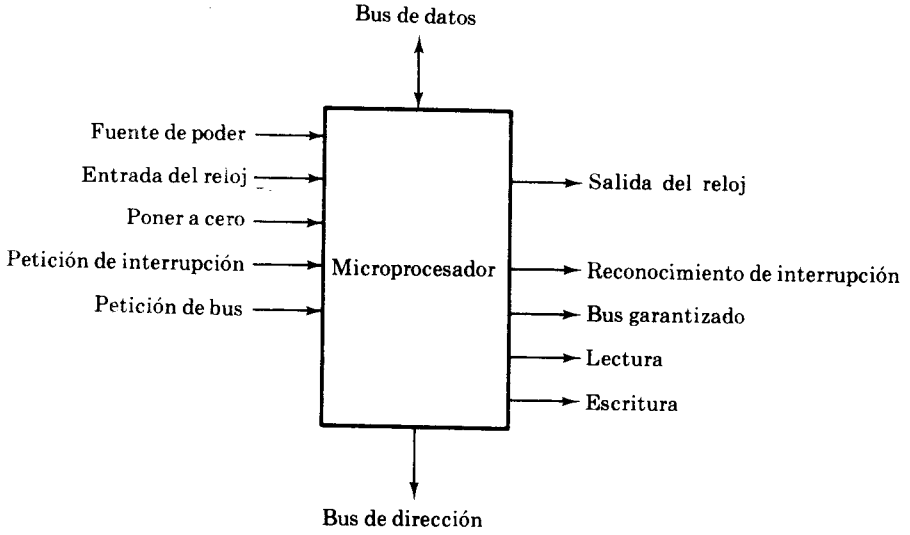


Figura 12-4 Señales de control en un microprocesador

el reloj de salida lo produce el generador de reloj en vez del microprocesador en sí. Algunas unidades generan el pulso de reloj dentro de sí, pero requieren un cristal externo o circuito para controlar la frecuencia del reloj. Los pulsos de reloj son usados por los módulos externos para sincronizar sus operaciones con las operaciones del microprocesador.

El terminal de puesta a cero o *reset* se usa para reposicionar o iniciar el microprocesador después de haber activado la potencia o en cualquier momento en que el usuario quiera comenzar el proceso desde el principio. El efecto de la señal de puesta a cero es iniciar el microprocesador, forzando una dirección dada al contador del programa. El programa comienza la ejecución con la primera instrucción en esa dirección. La manera más simple de iniciar una puesta a cero es borrar el contador del programa y comenzar el programa desde la dirección cero. Algunos microprocesadores responden a la señal de puesta a cero trasfiriendo el contenido de un lugar de memoria específico al contador del programa. El diseñador debe almacenar la dirección de comienzo del programa en el lugar de memoria adoptado.

La requisición de *interrrupción* (interrupt) al microprocesador, viene típicamente de un módulo de interconexión para informar al microprocesador que esté listo para trasferir la información. Cuando el microprocesador recibe una requisición de interrupción, suspende la ejecución del programa corriente y se bifurca a un programa que sirve de módulo de interconexión. Al completar la rutina de servicio, el computador regresa al programa previo. La facilidad de interrupción se incluye para producir un cambio en la secuencia del programa como resultado de las condiciones externas. El concepto de interrupción y el método de responder a una requisición de interrupción se discute en la Sección 12-5.

El terminal de entrada de *bus-request* (requisición del bus) es una requisición al microprocesador para suspender su operación y llevar todos los buses a su estado de mayor impedancia. Una vez reconocida la requisición, el microprocesador responde habilitando la línea de salida de control de *garantía de bus* (*bus-granted*). Así, cuando un dispositivo externo desea trasferir la información directamente a la memoria, éste solicita que el microprocesador abandone el control del bus común. Una vez que el bus sea inhabilitado por el microprocesador el dispositivo que originó la requisición toma control sobre el bus de direcciones y datos para conducir las trasferencias de memoria sin la intervención del procesador. Esta característica se llama *acceso directo de memoria* y se discute en la Sección 12-8.

Lectura y *escritura* son líneas de control que informan el componente seleccionado por el bus de direcciones de la dirección de la trasferencia esperada en el bus de datos. La línea de lectura informa a la unidad seleccionada que el bus de datos está en el modo de entrada y que el procesador aceptará datos del bus de datos. La línea de escritura indica que el procesador está en el modo de salida y que los datos válidos están disponibles en el bus de datos. Cuando los buses están inhabilitados, las dos líneas de control estarán en el estado de alta impedancia; así, la unidad externa que controla los buses puede especificar las operaciones de lectura y escritura.

Existen otras posibilidades para el control de los buses. El bus de direcciones puede ser controlado con una línea adicional para indicar si la dirección es para una palabra de memoria o para una unidad de intercone-

xión. Otra posibilidad es combinar las líneas de control de lectura y escritura en una línea que se denomina R/W. Cuando esta línea es 1 indica lectura y cuando es 0 indica escritura. Una segunda línea de control es necesaria para indicar cuándo una dirección válida está en el bus de direcciones de manera que los componentes externos respondan a la línea R/W solamente cuando se solicita con una dirección válida.

Las señales de control enumeradas en la Figura 12-4 constituyen un conjunto mínimo de funciones de control para un microprocesador. La mayoría de los microprocesadores tienen características de control adicionales para funciones especiales. Unidades diferentes pueden usar nombres numéricos diferentes para funciones de control idénticos y no necesariamente los nombres usados aquí.

Ejemplo de CPU

Para apreciar las tareas realizadas por un microprocesador, puede ser instructivo investigar la organización interna de una unidad típica. La Figura 12-5 muestra el diagrama de bloque de una unidad procesadora central encerrada dentro de una cápsula microprocesadora.* Externamente cuenta con un bus de datos bidireccional, un bus de direcciones y un número de líneas de control. Aquí se muestran solamente las líneas de control asociadas con la trasferencia en el bus. El bus de datos se designa por medio del símbolo DBUS y consiste de ocho líneas. La información contenida en las ocho líneas se llama *byte* u octeto, el cual constituye un nombre para denotar una *palabra de 8 bits*. El bus de direcciones designado por el símbolo ABUS, consiste de 16 líneas para especificar $2^{16} = 64K$ ($K = 1.024$) direcciones posibles. Así, el microprocesador es capaz de comunicarse con una unidad de memoria de 64K bytes.

Internamente, el microprocesador tiene seis registros procesadores demarcados B hasta G, un registro acumulador designado por la letra A y un registro temporal T. Estos registros son de 8 bits de longitud y pueden acumular un byte. El ALU opera con los datos almacenados en A y T y el resultado de la operación se trasfiere a A o a través de un bus interno a cualquiera de los seis registros procesadores. El registro de condición retiene el bit de condición de una operación tal como el arrastre final del ALU, el valor del bit de signo y la indicación de resultado cero.* El código de operación de una instrucción se trasfiere al registro de instrucción (IR), donde se decodifica para determinar la secuencia de microoperaciones necesaria para ejecutar la instrucción. El temporizador y control supervisa todas las operaciones internas en el CPU y las líneas de control externas en el microprocesador.

*Esta es similar al microprocesador 8080/85 excepto que los registros F y G se llaman H y L en el 8080/85.

*El registro de condición fue discutido en la Sección 9-7.

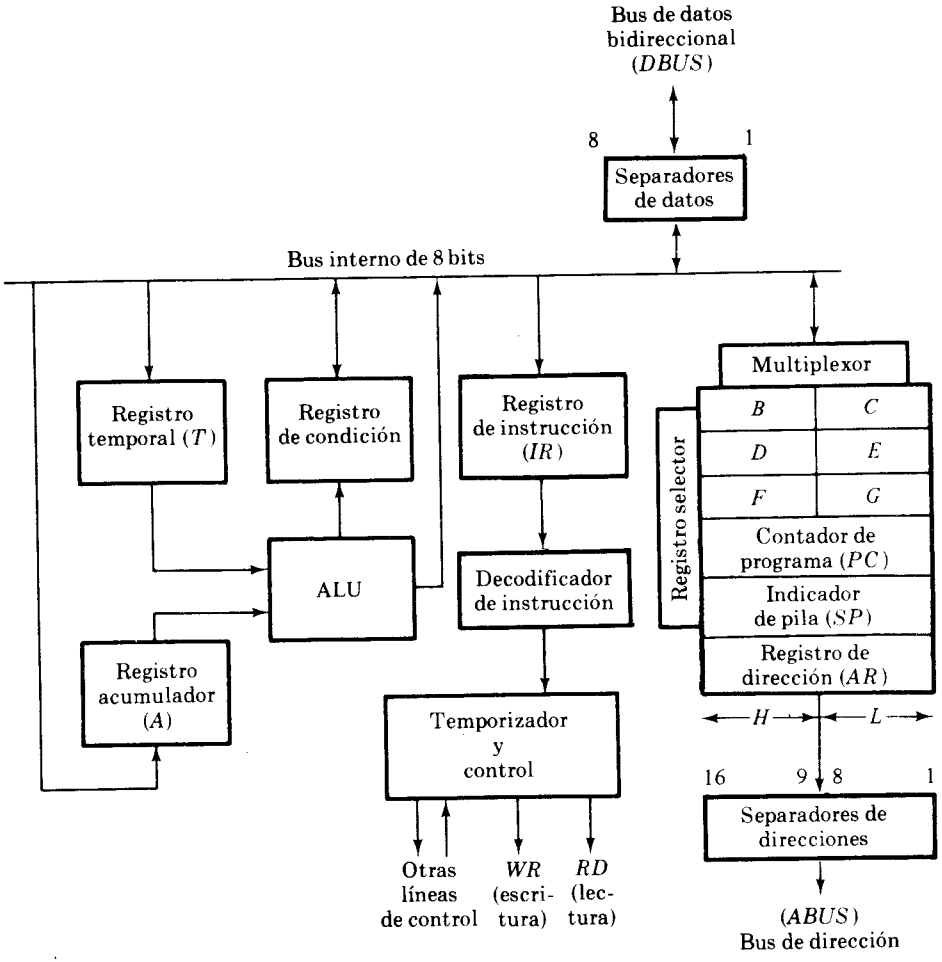


Figura 12-5 Diagrama de bloque del microprocesador

Los separadores de direcciones reciben la información de tres fuentes: el contador del programa (PC), el indicador de pila o stack pointer (SP) y el registro de direcciones (AR). El PC mantiene la dirección de memoria de la instrucción corriente del programa y se incrementa después de cada búsqueda de instrucción. El AR se usa para almacenamiento temporal de las direcciones que se leen de la memoria. Las funciones de estos dos registros serán clarificadas cuando se describan las secuencias de operaciones del CPU. SP se usa conjuntamente con una pila de memoria y su función se explica en la Sección 12-5. El bus de direcciones puede recibir información de direccionamiento de un par de registros procesadores. Se pueden formar tres pares para conformar una dirección de 16 bits. Estos se demarcan con los símbolos de registros combinados BC, DE y FG. Cada registro procesador contiene 8 bits y cuando se combina con el adyacente, conforma un par de registros de 16 bits. Es conveniente algunas veces dividir los tres

registros de 16 bits *PC*, *SP* y *AR* en dos partes. El símbolo *H* designa los 8 bits de mayor orden y el símbolo *L* los 8 bits de menor orden. Así *PC* (*L*) se refiere a los bits 1 a 8 del *PC* y *PC* (*H*) se refiere a los bits 9 a 16.

Ciclo de memoria

La unidad de memoria consiste de RAM y ROM. Esta se conecta al microprocesador por medio de los buses de direcciones y datos y el control de lectura y escritura. Esta se muestra esquemáticamente en la Figura 12-6. Un ciclo de memoria se define como el tiempo que se requiere para tener acceso a la memoria con el objeto de leer o escribir un byte.

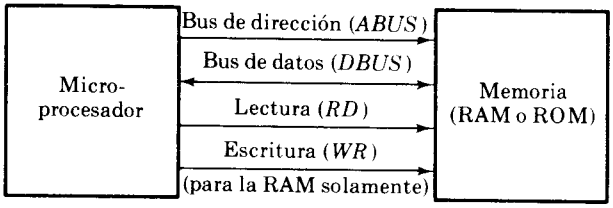


Figura 12-6 Comunicación entre el microprocesador y la memoria

En el ciclo de *lectura*, el microprocesador coloca una dirección en *ABUS* y habilita la línea de control *RD*. La memoria responde leyendo el byte y colocándolo en el *DBUS*. El microprocesador acepta el byte y lo trasfiere a un registro externo. Para expresar el ciclo de lectura simbólicamente se asume que la dirección viene del *AR* y el byte de datos se trasfiere al registro *A*:

$ABUS \leftarrow AR, RD \leftarrow 1$	dirección en el bus para lectura
$DBUS \leftarrow M[ABUS]$	la memoria lee el byte
$A \leftarrow DBUS, RD \leftarrow 0$	el byte se trasfiere a *A*

Primero, el microprocesador coloca la dirección de memoria en el *ABUS* e informa a la memoria que hay una dirección válida disponible para lectura. La memoria responde al *RD* leyendo el byte en la dirección dada por *ABUS* y colocándola en *DBUS*. El microprocesador trasfiere entonces el byte del *DBUS* a *A*. Al mismo tiempo la señal de control *RD* se inhabilita indicando el final de la trasferencia de memoria.

Las tres operaciones listadas anteriormente pueden combinarse en una sola proposición:

$$A \leftarrow M[AR]$$

Esta es la operación de lectura que trasfiere el byte de memoria direccionado por el *AR* al registro *A*.

En el ciclo de *escritura,* el microprocesador coloca una dirección en *ABUS* y un byte de datos en *DBUS*. Al mismo tiempo se habilita la línea *WR*. La memoria responde a *WR* escribiendo el byte del *DBUS* en un lugar de memoria especificado por la dirección en *ABUS*. Este proceso puede ser explicado simbólicamente:

$$ABUS \leftarrow AR,\ DBUS \leftarrow A,\ WR \leftarrow 1$$

$$M[ABUS] \leftarrow DBUS,\ WR \leftarrow 0$$

Este proceso establece que el contenido del registro *A* se trasfiera al byte de memoria en la dirección dada por *AR*. De nuevo es posible escribir esta operación con una proposición compuesta:

$$M[AR] \leftarrow A$$

Las trasferencias de memoria al microprocesador deben estar acordes con ciertas relaciones de tiempo que deben existir entre las señales de control y la información de los buses. Estas relaciones de tiempo son especificadas por formas de onda de tiempo que se incluyen en las unidades componentes con las especificaciones del producto. El intervalo de tiempo de un ciclo de memoria es una función de la frecuencia de reloj interna del microprocesador y el tiempo de acceso de la memoria. Una vez que el microprocesador envía la dirección, éste espera una respuesta dentro de un intervalo dado de tiempo. Una memoria capaz de responder dentro del intervalo de tiempo del procesador puede ser controlada directamente por el ciclo de memoria del microprocesador.

Si el microprocesador se comunica con una memoria lenta, podría tomar más tiempo el acceso de la memoria que el intervalo de tiempo permisible. Para poder usar memorias lentas el microprocesador debe ser capaz de demorar la trasferencia hasta que se complete el acceso de la memoria. Una forma es expandir el período del reloj del microprocesador es reduciendo la frecuencia del reloj para que se ajuste al tiempo de acceso de la memoria. Algunos microprocesadores vienen con un terminal de control especial, llamado *ready,* (listo) para permitir que la memoria coloque su propio tiempo de ciclo de memoria. Si el microprocesador no recibe la señal *ready* (listo) de la memoria, después de enviar la dirección, entrará en un estado de *espera* (wait), durante el tiempo en que la línea de *ready* (listo) esté en el estado 0. Cuando se complete el acceso de la memoria, la línea de ready pasa al estado 1 para indicar que la memoria está lista para una trasferencia específica.

Secuenciamiento del microprocesador

La secuencia y control en el microprocesador determina la secuencia de trasferencias a través de los buses internos y externos, el ALU y los registros procesadores. Durante el ciclo de búsqueda, el control lee un código de operación de la memoria y lo deposita en un registro de instrucción. La instrucción se decodifica y se traslada a actividades de procesamiento espe-

cíficas. Las referencias posteriores a la memoria dependen del código de operación decodificado. Asúmase que todos los códigos de operación consisten de ocho bits y se almacenan en un byte de memoria. Las operadores son también de un byte de longitud porque el bus de datos es de ocho bits de longitud. Una dirección se especifica con dos bytes o 16 bits. Considérese ahora tres instrucciones de suma con longitudes diferentes de formato.

1. *Sumar* B *a* A. Esta es una instrucción para sumar el contenido del registro *B* al contenido presente del acumulador. Toda la información necesaria para especificar la instrucción está contenida dentro de un código de operación de un byte.
2. *Sumar el operando inmediato a* A. Esta es una instrucción que suma un operando al contenido presente del acumulador. El byte del operando se coloca en la memoria siguiendo el byte del código de operación. Esta instrucción ocupa dos bytes de memoria.
3. *Sumar el operando especificado por una dirección a* A. Esta es una instrucción que suma un byte almacenado en alguna parte de la memoria al contenido presente del acumulador. La dirección del operando se coloca en la memoria siguiendo el byte del código de operación. Esta instrucción ocupa tres bytes de memoria, ya que las direcciones en sí mismas ocupan dos bytes.

El formato y la función de las tres instrucciones se sumarizan en la Tabla 12-1. Cada instrucción tiene al menos un byte para el código de operación. La unidad de control se diseña para reconocer el número de bytes en una instrucción particular del código de operación decodificado del primer byte.

La representación de memoria de las tres instrucciones se ilustra en la Figura 12-7. La primera instrucción se asume ubicada en el lugar 81 con un código de operación de 8 bits asignados arbitrariamente. Las otras dos instrucciones ocupan dos o tres bytes respectivamente. La dirección de la primera instrucción es 260 y se determina del número binario de 16 bits en los lugares 85 y 86:

$$(00000001\ 00000100)_2 = (260)_{10}$$

El operando para esta instrucción se muestra localizado en la memoria en el lugar 260. En una aplicación típica, las tres instrucciones residirán nor-

Tabla 12-1 Tres instrucciones típicas de un microprocesador

Instrucción	Byte 1	Byte 2	Byte 3	Función
Sumar *B* y *A*	Código de operación	—	—	$A \leftarrow A + B$
Sumar el operando en turno y *A*	Código de operación	Operando	—	$A \leftarrow A$ + byte 2
Sumar el operando especificado por una dirección y *A*	Código de operación	Mitad de mayor orden de la dirección	Mitad de menor orden de la dirección	$A \leftarrow A + M[\text{dirección}]$

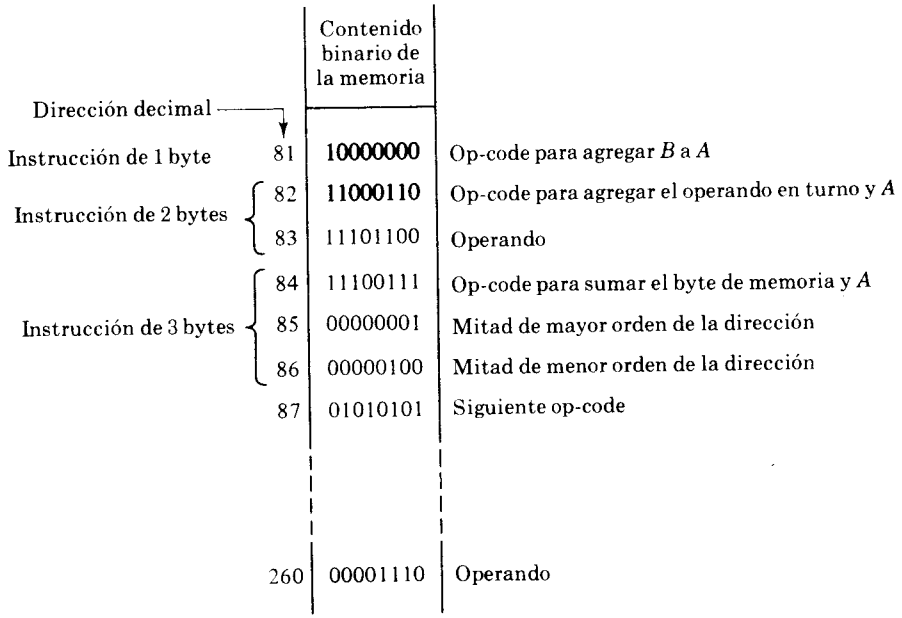


Figura 12-7 Representación de la memoria de tres instrucciones

malmente en la ROM, mientras que el operando en el lugar 260 estará en la RAM. Este operando debe residir en la RAM porque debe asumirse que su valor está sujeto a cambio durante el cálculo. De otra forma si el valor del operando no cambia, no habrá necesidad de asociarlo con una dirección.

Refiriéndose a la Figura 12-5 para los nombres de los registros y buses, se puede hacer una lista de la secuencia de operaciones necesarias para procesar cada instrucción. Se asume que el contador del programa contiene inicialmente 81.

Sumar B *a* A:

$IR \leftarrow M[PC], PC \leftarrow PC + 1$	leer el código de operación
$T \leftarrow B$	trasferir B a T
$A \leftarrow A + T$	sumar T a A

La primera línea representa el ciclo de búsqueda para leer el código de operación al registro de sustracción. La operación decodificada especifica un registro procesador; de manera que el contenido de B se trasfiere a T y la operación de suma se realiza en el ALU. Nótese que el PC se ha incrementado y contiene ahora el número 82.

La instrucción de un byte se ejecuta con un ciclo de memoria porque todos los operandos residen en los registros del procesador. Si un operando reside en la memoria, es necesario accesar la memoria para leer el operando.

Sumar el operando inmediato a A:

$IR \leftarrow M[PC], PC \leftarrow PC + 1$	leer el código de operación
$T \leftarrow M[PC], PC \leftarrow PC + 1$	leer el operando
$A \leftarrow A + T$	sumar el operando a A

La primera línea representa el ciclo de búsqueda una vez más. El PC se incrementa para que contenga la dirección 83. En esta dirección se lee el operando de la memoria y se coloca en T para ejecutar la suma en el ALU.

Si la instrucción contiene la dirección del operando, el microprocesador debe pasar por cuatro ciclos de memoria para ejecutar la instrucción.

Sumar a A *el operando especificado por una dirección:*

$IR \leftarrow M[PC], PC \leftarrow PC + 1$	leer el código de operación
$AR(H) \leftarrow M[PC], PC \leftarrow PC + 1$	leer el primer byte de la dirección
$AR(L) \leftarrow M[PC], PC \leftarrow PC + 1$	leer el segundo byte de la dirección
$T \leftarrow M[AR]$	leer el operando
$A \leftarrow A + T$	sumar el operando a A

La parte de la dirección de la instrucción se almacena temporalmente en el registro de dirección (AR). La dirección de 16 bits formada en el AR se usa entonces para leer el operando.

No se requiere un gran número de ciclos de memoria en los microprocesadores porque consumen gran cantidad de tiempo de proceso. Este es uno de los factores limitantes de la velocidad de los microprocesadores de 8 bits con direcciones de 16 bits. El número de accesos a la memoria puede ser reducido si se usa el bus de datos de 16 bits. Los microprocesadores de 16 bits requieren menos referencias a la memoria comparados con los microprocesadores de 8 bits. Aunque se ha escogido describir la operación de un microprocesador de 8 bits, la operación con un bus de datos de 16 bits podría ser similar, tomando en consideración las diferencias en longitudes de las palabras usadas por los registros del procesador y las palabras de memoria.

12-4 INSTRUCCIONES Y MODOS DE DIRECCIONAMIENTO

La estructura lógica de los microprocesadores se describe en los manuales de referencia suministrados por el fabricante. Un manual para un microprocesador particular describe la organización interna en el CPU, la función de todos los terminales de entrada y salida y los registros procesadores disponibles desde el punto de vista del usuario. El manual describe todas las instrucciones disponibles en el computador y explica sus funciones. Demues-

tra también cómo los bits de condición son afectados por cada instrucción. El código interno para cada instrucción se lista en binario, octal y hexadecimal. En la mayoría de los casos se adopta un código equivalente octal o hexadecimal porque estos códigos necesitan menos dígitos que la representación binaria. Cuando se escribe un programa para un computador se asigna a cada instrucción un nombre simbólico para identificarlo.

Los nombres simbólicos y códigos asignados a las instrucciones usadas en un microprocesador son diferentes de los nombres y códigos usados en un microprocesador diferente aun para instrucciones similares. Por esta razón, el usuario debe estudiar y recordar el conjunto de instrucciones y sus nombres simbólicos cada vez que se usa un microprocesador diferente. Aunque el grupo de instrucciones de diferentes microprocesadores difieren de uno a otro, hay ciertas instrucciones que realizan operaciones básicas y que son incluidas en todos los microprocesadores.

Conjunto básico de instrucciones de un microprocesador

Las instrucciones del microprocesador pueden clasificarse en tres tipos diferentes.

1. Instrucciones de *trasferencia* que mueven datos entre registros, palabras de memoria y registros de interconexión sin cambiar el contenido de la información binaria.
2. Instrucciones de *operación* que realizan operaciones con los datos almacenados en los registros o palabras de memoria.
3. Instrucciones de *control* usados para probar el estado de las codificaciones en los registros y causar un cambio en la secuencia del programa dependiendo de los resultados.

El conjunto de instrucciones de un microprocesador particular especifica las operaciones de trasferencia entre registros y decisiones de control que están presentes en el sistema del microcomputador. Un programa específico para un microcomputador es equivalente a especificar la secuencia de operaciones para un sistema digital particular que sea configurado por el microcomputador.

Las instrucciones de tipo trasferencia en los microprocesadores son indicadas por diferentes nombres. Una instrucción de *movimiento* (move) causa una trasferencia de datos desde la fuente hasta su destino. La fuente o el destino puede ser un registro procesador o un lugar de memoria. Las instrucciones de *carga* (load) y *almacenar* (store) son similares en la instrucción de movimiento excepto que se refieren normalmente a trasferencias a la memoria y al acumulador y viceversa. La instrucción de *intercambio* (exchange) cambia la información entre dos registros o entre un registro y una palabra de memoria. Las instrucciones de *insertar* (push) y *sacar* (pop) trasfieren datos entre los registros procesadores y la pila de memoria. Las instrucciones de *entrada* y *salida* trasfieren datos entre los registros procesadores y los registros de interconexión.

Las instrucciones de tipo operativo ejecutan operaciones aritméticas y de desplazamiento entre los registros procesadores o palabras de memoria. Ellas ponen a cero, a uno o complementan los bits de condición o bits indicadores. Las instrucciones de operación típicas son sumar, restar, AND, OR, complementar y poner a uno el bit de arrastre. La mayoría de las instrucciones de tipo operativo cambian también los bits de condición en el registro de condición del procesador.

Las instrucciones de tipo control tienen características para tomar decisiones y cambiar el camino tomado por el programa cuando se ejecuta en el computador. Las instrucciones se almacenan en lugares de memoria consecutivos y se ejecutan una detrás de otra en secuencia. el programador agrega una instrucción de control cada vez que el control debe ser trasferido a una instrucción que esté fuera de la secuencia normal. Las instrucciones de control pueden ser condicionales o incondicionales. Una instrucción de control condicional causa una bifurcación de la secuencia de programa normal solamente cuando se detecta una condición de estado específica. Una instrucción de control incondicional causa una bifurcación incondicional. La bifurcación de la secuencia del programa normal se logra cambiando el contador del programa de manera que éste contenga la dirección de la instrucción que esté próxima a ejecutarse.

Hay tres tipos de instrucciones de control y cada tipo puede ser condicional o incondicional:

1. Instrucciones de salto o bifurcación.
2. Instrucciones de llamado y regreso a la subrutina.
3. Instrucciones de omisión.

Las palabras *saltar* o *bifurcar* se usan igualmente para implicar lo mismo y algunas veces se usan para denotar modos diferentes de direccionamiento. Estas instrucciones están asociadas con una dirección que especifica dónde se debe hacer el salto o la bifurcación. Las instrucciones de subrutina de llamado y de regreso se explicarán en la siguiente sección conjuntamente con la pila de memoria. Una instrucción de salto u omisión (skip) es la que evita la siguiente instrucción en secuencia. Al colocar una instrucción de bifurcación incondicional enseguida de una instrucción de omisión o salto, es posible hacer una bifurcación a uno de dos lugares posibles, dependiendo del valor especificado de la condición del bit de status.

Instrucciones para el microprocesador

El número de instrucciones diferentes en un microprocesador particular puede variar entre 50 y 250. Estas instrucciones deben ser estudiadas y memorizadas por el usuario quien escribe los programas para el microcomputador. Una lista parcial de instrucciones formuladas para el microprocesador de la Figura 12-5 se presenta en la Tabla 12-2. Estas instrucciones se

Tabla 12-2 Lista parcial de instrucciones para el microprocesador

Código hexa-decimal	Símbolo de la instrucción		Descripción	Función*
78	MOV	A, B	Mover B hacia A	$A \leftarrow B$
3E	MVI	A, D8	Mover el operando inmediato a A	$A \leftarrow D8$
7E	MOV	A, FG	Mover hacia A con registro indirecto	$A \leftarrow M[FG]$
77	MOV	FG, A	Mover A con registro indirecto	$M[FG] \leftarrow A$
3A	LDA	AD16	Cargar A directamente	$A \leftarrow M[AD16]$
32	STA	AD16	Almacenar A directamente	$M[AD16] \leftarrow A$
01	LXI	FG, D16	Cargar el par de registros inmediatamente	$FG \leftarrow D16$
80	ADD	B	Sumar B a A	$A \leftarrow A + B$
C6	ADI	D8	Sumar el operando inmediato a A	$A \leftarrow A + D8$
86	ADD	FG	Sumar a A con registro indirecto	$A \leftarrow A + M[FG]$
90	SUB	B	Sustraer B de A	$A \leftarrow A - B$
A0	ANA	B	AND B y A	$A \leftarrow A \wedge B$
B0	ORA	B	OR B y A	$A \leftarrow A \vee B$
04	INR	B	Incrementar B	$B \leftarrow B + 1$
05	DCR	B	Decrementar B	$B \leftarrow B - 1$
03	INX	BC	Incrementar el par de registros BC	$BC \leftarrow BC + 1$
0B	DCX	BC	Decrementar el par de registros BC	$BC \leftarrow BC - 1$
2F	CMA		Complementar A	$A \leftarrow \overline{A}$
07	RLC		Rotar A a la izquierda con arrastre	$A \leftarrow$ clc A
0F	RRC		Rotar A a la derecha con arrastre	$A \leftarrow$ crc A
37	STC		Poner a 1 el bit de arrastre	$C \leftarrow 1$
C3	JMP	AD16	Saltar incondicionalmente	$PC \leftarrow AD16$
DA	JC	AD16	Saltar si existe arrastre	Si $(C = 1)$ entonces $(PC \leftarrow AD16)$
C2	JNZ	AD16	Saltar si existe valor diferente a cero	Si $(Z = 0)$ entonces $(PC \leftarrow AD16)$
CD	CALL	AD16	Llamar subrutina	$Pila \leftarrow PC$, $PC \leftarrow AD16$
C9	RET		Regreso de la subrutina	$PC \leftarrow pila$
76	HLT		Detener el procesador	

*A = registro acumulador; B = registro B; FG = par de registros F y G; BC = par de registros B y C; $D8$ = operando de datos de 8 bits (1 byte); $D16$ = operando de datos de 16 bits (2 bytes); $AD16$ = dirección de 16 bits (2 bytes).

dividen en tres secciones para dar ejemplos de instrucciones del tipo de trasferencia, operación y control.

El código hexadecimal listado en la tabla es un número de 2 dígitos equivalente al código de operación de 8 bits asignado a la instrucción. (La representación de 4 bits equivalente para los 16 dígitos hexadecimales está dada en la Tabla 1-1.) El nombre simbólico de cada instrucción es una designación de 2 a 4 letras seguidas de uno o dos símbolos de un registro, un operando o una dirección de memoria. La columna de descripción explica la instrucción en palabras y la columna de función define la instrucción precisamente con una proposición de trasferencia entre registros. Nótese que las instrucciones de computador especifican macrooperaciones para él mismo y pueden ser simbolizados con proposiciones apropiadas por el método de trasferencia entre registros. Sin embargo, por razones prácticas, las instrucciones de computador se escriben con símbolos específicos como en la segunda columna de la tabla. Estos símbolos especiales son asignados por el fabricante del computador y tienden a ser diferentes en los diferentes computadores.

Las primeras cuatro instrucciones de la Tabla 12-2 son instrucciones de movimiento y trasferencia de información de una fuente a un destino dado. Las siguientes son instrucciones de carga y almacenaje que logran un objetivo similar. Un número representativo de instrucciones del tipo operativo se listan en la segunda parte de la tabla. En la última sección se da la lista de varias instrucciones de control.

La instrucción de mover con registro indirecto:

MOV A, FG

simboliza la operación de trasferencia entre registros $A \leftarrow M[FG]$. Esta trasfiere al registro A el byte de memoria, cuya dirección está en el par de registros FG. Esta es llamada instrucción indirecta de registro ya que el par de registros FG especifican la dirección del operando en vez del operando en sí.

La instrucción de carga inmediata:

LXI FG, D16

simboliza la operación de trasferencia entre registros $FG \leftarrow D16$ donde $D16$ es un número de 2 bytes que puede representar una dirección. Esta instrucción puede ser usada para trasferir una dirección al par de registros FG. Cuando se usa de esta manera el par de registros FG constituyen un *contador de datos* o un *indicador* que señala una dirección de memoria donde se almacena el operando. FG puede ser incrementado con la instrucción de incremento del par de registros:

INX FG

la cual simboliza la operación de trasferencia entre registros $FG \leftarrow FG + 1$. De esta manera, el contador de datos o indicador puede ser incrementado para indicar las direcciones consecutivas en la memoria donde el programador almacena una tabla de bytes de datos consecutivos.

Las instrucciones de operación contienen las operaciones comunes aritméticas, lógicas y de desplazamiento. Nótese que hay más instrucciones del mismo tipo que pueden ser formuladas si se especifica uno de los cinco registros procesadores *C, D, E, F* o *G* en vez del registro específico *B*. Similarmente, una instrucción que especifica un par de registros puede ser duplicada usando uno cualquiera de los tres pares de registros posibles *BE, DE* o *FG*.

Las últimas seis instrucciones en la tabla son instrucciones de control. Las instrucciones de salto y llamado necesitan una dirección de 16 bits simbolizada *AD*16. Las direcciones de regreso o paro son instrucciones de un byte. Aquellas que incluyen el símbolo *AD*16 o *D*16 son instrucciones de tres bytes y aquellas que usan el símbolo *D*8 son instrucciones de dos bytes. Todas las demás son instrucciones de un byte especifiquen o no un registro.

La mejor manera de apreciar un conjunto de instrucciones de un computador es escribir programas que realicen tareas de procesamiento de datos significativos. Los programas escritos para el sistema de microcomputador requieren el mismo razonamiento lógico al escribir microprogramas para un sistema digital como se vio en el ejemplo del Capítulo 10.

Modos de direccionamiento

El código de operación de una instrucción especifica la operación que va a ser ejecutada después de haberse leído de la memoria y colocado en la unidad de control del CPU. La unidad de control debe saber dónde encontrar al operando con el cual se va a ejecutar la operación. Los operandos pueden estar localizados en los registros de proceso en palabras de memoria o en los registros de interconexión. La forma como son determinados los operandos durante la ejecución del programa se determina a partir del *modo de direccionamiento* de la instrucción. En computadores grandes, el modo de direccionamiento de una instrucción se especifica con un código binario de la misma forma como se especifica el código de operación. En los microprocesadores de 8 bits, el primer byte de una instrucción es un código binario combinado que especifica la operación y el modo de la instrucción. Una vez colocado este byte en el registro de instrucción, durante el ciclo de búsqueda es interpretado por el control para determinar no solamente la operación que debe ser ejecutada sino también la forma como se van a localizar los operandos.

En la Tabla 12-1 se puede encontrar un ejemplo de tres modos de direccionamiento para la misma operación. La tabla define tres tipos de modos de direccionamiento para la instrucción *sumar a A*. La operación puede referirse a un registro, a un operando inmediato o a un operando específico para una dirección de memoria, si ésta se especifica de modo diferente. Un computador puede usar una gran variedad de modos de direccionamiento para la misma operación para emplear diferentes maneras de localizar operandos. Para el usuario sin experiencia, la variedad de modos de direccionamiento en algunos computadores puede parecer excesivamente complicada. Sin embargo, la disponibilidad de diferentes esquemas de programación

dan al programador con experiencia la flexibilidad de escribir programas que son más eficientes con respecto al número de instrucciones y tiempo de ejecución.

Se han discutido algunos modos de direccionamiento en los ejemplos anteriores y se sumarizan aquí como referencia.

Modo implícito: En este modo se especifica el operando implícitamente en la definición de la instrucción. Las instrucciones de este tipo son instrucciones de 1 byte. Por ejemplo, la instrucción "complementar el acumulador" es una instrucción en modo implicado porque el operando en el registro acumulador está implícito en la definición de la instrucción.

Modo de registro: En este modo los operandos están en los registros que residen dentro del CPU. Las instrucciones del modo de registro son instrucciones de 1 byte y puede ser ejecutadas dentro del CPU sin necesidad de hacer referencia a las memorias para los operandos.

Modo indirecto de registro: En este modo la instrucción especifica un registro o un par de registros en el procesador cuyo contenido da la dirección del operando en la memoria. Este modo usa instrucciones de 1 byte aunque el operando esté en la memoria. Antes de usar una instrucción de modo indirecto de registro, el programador debe asegurarse que la dirección del operando se coloque en el registro procesador con una instrucción previa de tipo trasferencia. Una referencia al registro es equivalente a especificar una dirección de memoria.

Modo inmediato: En este modo el operando se especifica en la dirección en sí. En un microprocesador de 8 bits se coloca el operando en la memoria inmediatamente después del byte del código de operación. Una instrucción de modo inmediato que tiene un operando de 8 bits es una instrucción de 2 bytes. Una con un operando de 16 bits es una instrucción de 3 bytes.

Modo de direccionamiento directo: En este modo el operando reside en la memoria y su dirección está dada directamente en la parte de dirección de la instrucción. Una instrucción directa consiste de tres bytes en un microprocesador de 8 bits con direcciones de 16 bits. En computadores con palabras de memoria mayores, la parte de dirección se combina con la operación y los bits del código de modo para unir toda la instrucción en una palabra de memoria. La mayoría de las instrucciones de modo directo asumen que los otros operandos residen en los registros procesadores. Si hay más de un operando que resida en la memoria, la instrucción debe incluir direcciones adicionales para especificar sus posiciones.

Algunos microprocesadores de 8 bits con direcciones de 16 bits tienen modos de direccionamiento directo que requieren solamente un byte para especificar una dirección. Tales microprocesadores dividen los 2^{16} bytes de memoria en bloques llamados *páginas*. A cada página se le asigna usual-

mente 256 bytes de espacio de memoria consecutiva. Una página en la memoria se especifica con los ocho bits de mayor orden de una dirección. Los 8 bits de menor orden dan el byte dentro de la página. Así, una memoria de 64K puede dividirse en 256 páginas de 256 bytes cada una. La primera página se llama página 0 y la última página 255. Por medio del esquema de páginas es posible desarrollar algunas variaciones en el modo directo de direccionamiento.

Direccionamiento de la página cero: Este es similar al modo de direccionamiento directo excepto que la parte de la dirección de la instrucción contiene solamente 1 byte. Esta es una instrucción de 2 bytes con un segundo byte especificando los ocho bits de menor orden de la memoria de direcciones. Los ocho bits de mayor orden de la dirección se asumen como ceros. Esto restringe el rango de las direcciones a los 256 bytes menores de memoria (0-255) los cuales definen la página 0.

Direccionamiento de página presente: Este modo asume que el operando reside en la memoria dentro de la misma página de memoria que la instrucción que la usa. Como el contador del programa retiene siempre la dirección de la siguiente instrucción, sus ocho bits de mayor orden contienen también el número de página presente. Este modo de direccionamiento usa instrucciones de 2 bytes con una parte de dirección de 8 bits. La dirección del operando se obtiene del número de página encadenado con la parte de dirección de la instrucción. La dirección de 16 bits del operando se calcula a partir de:

$$PC(H) + AD8$$

donde $PC(H)$ denota los ocho bits de mayor orden del PC y $AD8$ los 8 bits de dirección de la instrucción. El resultado es una dirección de 16 bits con el $PC(H)$ dando los primeros 8 bits y $AD8$ los 8 bits restantes.

Direccionamiento relativo: Este es similar al modo de direccionamiento de página presente excepto de que no es sensible a los límites de las páginas. Una instrucción de modo relativo es una instrucción de 2 bytes con el segundo byte que especifica un número con signo en el rango entre -128 y $+127$. Esto se logra representando el número en la forma de signo complemento de 2. La dirección de 16 bits del operando se calcula agregando el contenido de 16 bits disponibles al presente en el contador del programa a la dirección de 8 bits con signo en la instrucción. Si esta última se denota como $AD8$ el cómputo de la dirección puede simbolizarse como:

$$PC + AD8$$

Esto requiere que el operando (o el lugar donde la instrucción relativa de bifurcación trasfiere el control) esté entre 127 y -128 bytes separado de la dirección de la siguiente instrucción. Los límites de la página no tienen ninguna consecuencia en el modo relativo porque todos los 16 bits del contador del programa se usan en el cálculo.

La parte de la dirección de una instrucción se usa por la unidad de control en el CPU para obtener el operando a partir de la memoria. Algunas veces esta dirección es la del operando, pero otras es una dirección de la cual se calcula la del operando. Los computadores usan otros modos de direccionamiento para calcular la dirección de un operando. Para distinguir entre las diferentes direcciones que intervienen en el cálculo, se debe distinguir entre las direcciones dadas en la instrucción y la dirección actual usada por el control cuando se ejecuta la instrucción. La dirección del operando o la dirección donde se bifurque el control en respuesta a un salto, una bifurcación o una instrucción de llamado, se denomina *dirección efectiva*. En la instrucción de modo directo, la dirección efectiva es igual a la parte de dirección de la instrucción. En el modo relativo la dirección efectiva se calcula a partir del valor en el PC más la parte de la dirección de la instrucción.

El cálculo de la dirección efectiva para los últimos cuatro modos de direccionamiento discutidos anteriormente se listan en la Tabla 12-3. En la tabla se da la lista de otros cinco modos de direccionamiento encontrados comúnmente en los microprocesadores (y en computadores de gran tamaño). El símbolo $AD16$ denota una dirección de 2 bytes y $AD8$ denota una dirección de 1 byte. PC es el contador del programa y XR es un registro índice. XR es un registro del CPU usado en muchos computadores para almacenar direcciones. La dirección almacenada en XR puede ser referenciada con una instrucción de modo indicado. Una instrucción se coloca inicialmente en el XR por medio de una instrucción del tipo trasferencia. El cálculo de la dirección efectiva en cada modo se especifica en la tabla con una expresión de cómputo del registro. La dirección efectiva de cómputo se usa para accesar la memoria a fin de leer un operando o convertirse en la dirección de bifurcación en una instrucción del tipo de control. Los otros modos de direccionamiento listados en la tabla se explican a continuación.

Tabla 12-3 Cálculo de la dirección efectiva para varios modos de direccionamiento

Modo de direccionamiento	Dirección efectiva	Comentarios
Directo	$AD16$	Parte de la dirección de 16 bits de la instrucción
Página cero	$AD8$	Parte de la dirección de 8 bits de la instrucción
Página presente	$PC(H) + AD8$	Los 8 bits de mayor orden del PC encadenados en $AD8$
Relativo	$PC + AD8$	Contenido del PC más $AD8$ con signo
Indexado	$XR + AD16$	Contenido del XR más $AD16$
Registro base	$XR + AD8$	Contenido de XR más $AD8$
Indirecto	$M[AD16]$	Dirección almacenada en el lugar dado por $AD16$
Indirecto-indexado	$M[XR + AD8]$	Dirección almacenada en el lugar $(XR + AD8)$
Indexado-indirecto	$M[AD8] + XR$	Dirección almacenada en el lugar $AD8$ más el contenido de XR