



ARQUITECTURA DE COMPUTADORAS

TERCERA EDICION



PEARSON

Prentice
Hall

M. Morris Mano

<i>Procedimiento de diseño</i>	36
Problemas	38
Referencias	41
CAPÍTULO DOS	
Componentes digitales	43
2-1 Circuitos integrados	43
2-2 Decodificadores	45
<i>Decodificador de compuerta NAND</i>	<u>47</u>
<i>Expansión del decodificador</i>	<u>48</u>
<i>Codificadores</i>	<u>49</u>
2-3 Multiplexores	50
2-4 Registros	53
<i>Registro con carga en paralela</i>	54
2-5 Registros de corrimiento	55
<i>Registro de corrimiento bidireccional con carga paralela</i>	56
2-6 Contadores binarios	59
<i>Contador binario con carga en paralelo</i>	61
2-7 Unidad de memoria	63
<i>Memoria de acceso aleatorio</i>	63
<i>Memoria sólo de lectura</i>	65
<i>Tipos de ROM</i>	66
Problemas	67
Referencias	69
CAPÍTULO TRES	
Representación de datos	71
3-1 Tipos de datos	71
<i>Sistemas numéricos</i>	72
<i>Números octales y hexadecimales</i>	74
<i>Representación decimal</i>	76
<i>Representación alfanumérica</i>	78
3-2 Complementos	78
<i>Complemento (r-1)</i>	79
<i>Complemento a (r)</i>	80

CAPÍTULO SEIS		
Programación básica de la computadora	183	
<hr/>		
6-1	Introducción	183
6-2	Lenguaje de máquina	184
6-3	Lenguaje ensamblador	189
	<i>Reglas del lenguaje</i>	189
	<i>Un ejemplo</i>	191
	<i>Traducción a binario</i>	191
6-4	El ensamblador	194
	<i>Representación de un programa simbólico en la memoria</i>	194
	<i>Primera pasada</i>	195
	<i>Segunda pasada</i>	197
6-5	Ciclos del programa	200
6-6	Programación de operaciones aritméticas y lógicas	203
	<i>Programa de multiplicación</i>	205
	<i>Suma de doble precisión</i>	206
	<i>Operaciones lógicas</i>	207
	<i>Operaciones de corrimiento</i>	208
6-7	Subrutinas	209
	<i>Parámetros de subrutina y enlace de datos</i>	211
6-8	Programación de entrada-salida	214
	<i>Manipulación de caracteres</i>	215
	<i>Interrupción del programa</i>	217
	Problemas	220
	Referencias	223
<hr/>		
CAPÍTULO SIETE		
Control microprogramado	225	
<hr/>		
7-1	Memoria de control	225
7-2	Secuencia de la dirección	228
	<i>Transferencia condicional</i>	231
	<i>Mapeo de las instrucciones</i>	231
	<i>Subrutinas</i>	232
7-3	Ejemplo de un microprograma	233
	<i>Configuración de la computadora</i>	233
	<i>Formato de las microinstrucciones</i>	235
	<i>Microinstrucciones simbólicas</i>	238

<i>La rutina de búsqueda</i>	239
<i>Microprograma simbólico</i>	240
<i>Microprograma binario</i>	243
7-4 Diseño de la unidad de control	244
<i>Secuenciador de microprograma</i>	246
Problemas	249
Referencias	253
 <hr/>	
CAPÍTULO OCHO	
Unidad central de procesamiento	
	255
8-1 Introducción	255
8-2 Organización general de los registros	256
<i>Palabra de control</i>	258
<i>Ejemplos de microoperaciones</i>	259
8-3 Organización de una pila	261
<i>Pila de registro</i>	262
<i>Pila de memoria</i>	264
<i>Notación polaca inversa</i>	266
<i>Evaluación de las expresiones aritméticas</i>	268
8-4 Formatos de las instrucciones	270
<i>Instrucciones de tres direcciones</i>	273
<i>Instrucciones de dos direcciones</i>	273
<i>Instrucciones de una dirección</i>	274
<i>Instrucciones de cero direcciones</i>	274
<i>Instrucciones RISC</i>	275
8-5 Modos de direccionamiento	275
<i>Ejemplo numérico</i>	280
8-6 Transferencia y manipulación de los datos	282
<i>Instrucciones de transferencia de los datos</i>	283
<i>Instrucciones de manipulación de los datos</i>	284
<i>Instrucciones aritméticas</i>	285
<i>Instrucciones lógicas y de manipulación de bits</i>	286
<i>Instrucciones de corrimiento</i>	288
8-7 Control del programa	289
<i>Bits de condiciones de estado</i>	291
<i>Instrucciones de brinco condicional</i>	293
<i>Llamada y retorno de subrutina</i>	295
<i>Interrupción del programa</i>	297
<i>Tipos de interrupciones</i>	299
8-8 Computadora de conjunto de instrucciones reducido (RISC)	300

<i>Características CISC</i>	301
<i>Características RISC</i>	302
<i>Ventanas de registros traslapados</i>	303
<i>RISC I de Berkeley</i>	306
Problemas	310
Referencias	316
 CAPÍTULO NUEVE	
Paralelismo y procesamiento de vector	319
 9-1 Procesamiento paralelo	319
9-2 Arquitectura paralela	322
<i>Consideraciones generales</i>	324
9-3 Línea paralela aritmética	328
9-4 Línea paralela de instrucciones	331
<u>Ejemplo: Línea paralela de instrucciones de cuatro segmentos</u>	332
<i>Dependencia de los datos</i>	334
<i>Manejo de las instrucciones de transferencia de control</i>	335
9-5 Arquitectura paralela RISC	337
<u>Ejemplo: Línea paralela de instrucciones de tres segmentos</u>	338
<i>Carga pospuesta o retardada</i>	339
<i>Transferencia pospuesta o retardada</i>	340
9-6 Procesamiento vectorial	342
<i>Operaciones de vectores</i>	343
<i>Multiplicación de matriz</i>	344
<i>Memoria entrelazada</i>	346
<i>Supercomputadoras</i>	347
9-7 Arreglo de procesador SIMD	349
<i>Arreglo de procesador conectado</i>	349
<i>Arreglo de procesador SIMD</i>	350
Problemas	351
Referencias	353
 CAPÍTULO DIEZ	
Aritmética de computadoras	355
10-1 Introducción	355
10-2 Suma y resta	356

	Contenido	xv
13-4	Comunicación y sincronización entre procesadores	543
	<i>Sincronización entre procesadores</i>	545
	<i>Exclusión mutua con semáforo</i>	545
13-5	Coherencia de caché	547
	<i>Condiciones para incoherencia</i>	547
	<i>Soluciones al problema de coherencia de caché</i>	549
	Problemas	551
	Referencias	552

Índice [555](#)

Este libro trata de la arquitectura de computadoras y de la organización y diseño de computadoras. La arquitectura de computadoras se interesa por la estructura y desempeño de los diferentes módulos funcionales de la computadora y cómo interactúan para atender las necesidades de procesamiento del usuario. La organización de las computadoras estudia la manera en que se conectan los componentes de la circuitería para formar un sistema computacional. El diseño de computadoras analiza el desarrollo de la circuitería de computadoras, tomando en consideración un cierto conjunto de especificaciones.

El libro proporciona el conocimiento básico necesario para comprender la operación de la circuitería de computadoras digitales y cubre los tres temas asociados con la circuitería de computadoras. Los capítulos del 1 al 4 presentan los diferentes componentes digitales que se usan en la organización y diseño de computadoras digitales. Los capítulos del 5 al 7 muestran en detalle los pasos que debe recorrer un diseñador para preparar las bases de una computadora elemental. Los capítulos del 8 al 10 examinan la organización y arquitectura de la unidad de procesamiento central. Los capítulos 11 y 12 estudian la organización y la arquitectura de entrada-salida, y de la memoria. El capítulo 13 aclara el concepto de multiprocesamiento. El libro se refiere primero al material más sencillo y después aborda temas más avanzados. Por lo tanto, los primeros siete capítulos cubren el material necesario para el conocimiento básico de la organización, diseño y programación de una computadora digital simple. Los últimos seis capítulos presentan la organización y arquitectura de las unidades funcionales separadas de la computadora digital, enfatizando los temas más avanzados.

En esta tercera edición el material está organizado como en la segunda y muchos de los temas no cambian. Sin embargo, la tercera edición ofrece nuevos temas en relación con la segunda edición. Todos los capítulos, excepto dos (el 6 y el 10), se han revisado por completo para actualizar el material y hacer más clara su presentación. Se agregaron dos novedades: el capítulo 9, acerca del procesamiento por arquitectura paralela y vectores, y el capítulo 13, acerca de multiprocesadores. Dos secciones tratan acerca de la computadora que incluye un conjunto reducido de instrucciones (RISC). El capítulo 5 se revisó por completo para simplificar y hacer más claro el

EN ESTE CAPÍTULO

- 1-1 Computadoras digitales
- 1-2 Compuertas lógicas
- 1-3 Álgebra booleana
- 1-4 Simplificación por mapas
- 1-5 Circuitos combinatorios
- 1-6 Flip-flops
- 1-7 Circuitos secuenciales

1-1 Computadoras digitales

digital

La computadora digital es un sistema digital que ejecuta diversas tareas de computación. La palabra *digital* implica que la información en la computadora se representa por variables que toman un número limitado de valores discretos. Estos valores se procesan internamente por componentes que pueden mantener un número limitado de estados discretos. Los dígitos decimales 0, 1, 2, ..., 9, por ejemplo, proporcionan 10 valores discretos. Las primeras computadoras electrónicas digitales, desarrolladas a finales de los años 40, se usaron principalmente para cómputos numéricos. En este caso los elementos discretos son los dígitos. De esta aplicación ha surgido el término *computadora digital*. En la práctica, las computadoras digitales funcionan más confiablemente si sólo se usan dos estados. Por la restricción física de los componentes y porque la lógica humana tiende a ser binaria (por ejemplo, proposiciones de cierto o falso, sí o no), los componentes digitales que están restringidos a tomar valores discretos se restringen aún más a tomar sólo dos valores y se dice que son *binarios*.

bit

Las computadoras digitales emplean el sistema numérico binario, que tiene dos dígitos: 0 y 1. A un dígito binario se le llama *bit*. La información

nible necesario para la operación eficaz de la computadora. El software del sistema es una parte indispensable del sistema total de la computadora. Su función es compensar las diferencias que existen entre las necesidades del usuario y la capacidad del hardware.

hardware

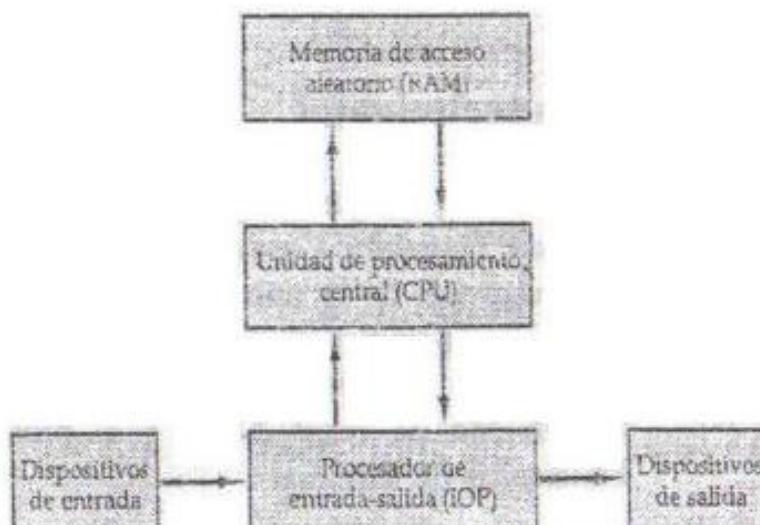
El *hardware* de la computadora se divide por lo general en tres grandes partes, como se muestra en la figura 1-1. La unidad central de procesamiento (CPU, central processing unit) contiene una unidad aritmética y lógica para la manipulación de datos, varios registros para almacenar los datos y circuitos de control para leer de la memoria y ejecutar instrucciones. La memoria de la computadora almacena las instrucciones y los datos. Se le llama memoria de acceso aleatorio (RAM, random access memory) por la CPU puede accesar cualquier parte de la memoria en forma aleatoria y recuperar la información binaria dentro de un intervalo fijo. El procesador de entrada/salida (IOP, input output processor) contiene circuitos electrónicos para comunicarse y controlar la transferencia de información entre la computadora y el mundo exterior. Los dispositivos de entrada y salida conectados a la computadora incluyen teclados, impresoras, terminales, unidades de discos magnéticos y otros dispositivos de comunicación.

Este libro proporciona el conocimiento básico para entender las operaciones del hardware de un sistema de computadora. El tema se considera a veces desde tres diversos puntos de vista, dependiendo del interés del investigador. Cuando se trata del hardware de la computadora, es costumbre distinguir entre lo que se refiere a la organización de computadora, su diseño y arquitectura.

organización de la computadora

La *organización de la computadora* se refiere a la manera en que los componentes operan y la forma en que se conectan para formar el sistema de la computadora. Se supone que los diversos componentes están en su lugar y la tarea es investigar la estructura organizacional para verificar que las partes de la computadora funcionen como se proponía.

Figura 1-1 Diagrama de bloque de una computadora digital.



CAPÍTULO DOS

Componentes digitales

EN ESTE CAPÍTULO

- 2-1 Circuitos integrados
- 2-2 Decodificadores
- 2-3 Multiplexores
- 2-4 Registros
- 2-5 Registros de corrimiento
- 2-6 Contadores binarios
- 2-7 Unidad de memoria

2-1 Circuitos integrados

CI

Los circuitos digitales se construyen con circuitos integrados. Un circuito integrado (abreviado CI) es un pequeño cristal de silicio semiconductor, llamado *microcircuito*, que contiene los componentes electrónicos para las compuertas digitales. Las diversas compuertas se conectan dentro del microcircuito para formar los circuitos requeridos. El microcircuito se monta en un encapsulado de cerámica o de plástico y las conexiones se sueldan con finos alambres de oro a terminales externas para formar el circuito integrado. El número de terminales puede fluctuar de 14 en un encapsulado pequeño de CI hasta 100 o más en un encapsulado mayor. Cada CI tiene una designación numérica impresa en la superficie del paquete, para su identificación. Cada proveedor publica un manual de datos o un catálogo con la descripción exacta y la información necesaria acerca de los circuitos integrados que fabrica.

Conforme la tecnología de los circuitos integrados va mejorando, el número de compuertas que pueden fabricarse en un solo microcircuito ha aumentado considerablemente. La diferenciación entre los microcircuitos que tienen unas cuantas compuertas internas y aquellos que tienen cientos o

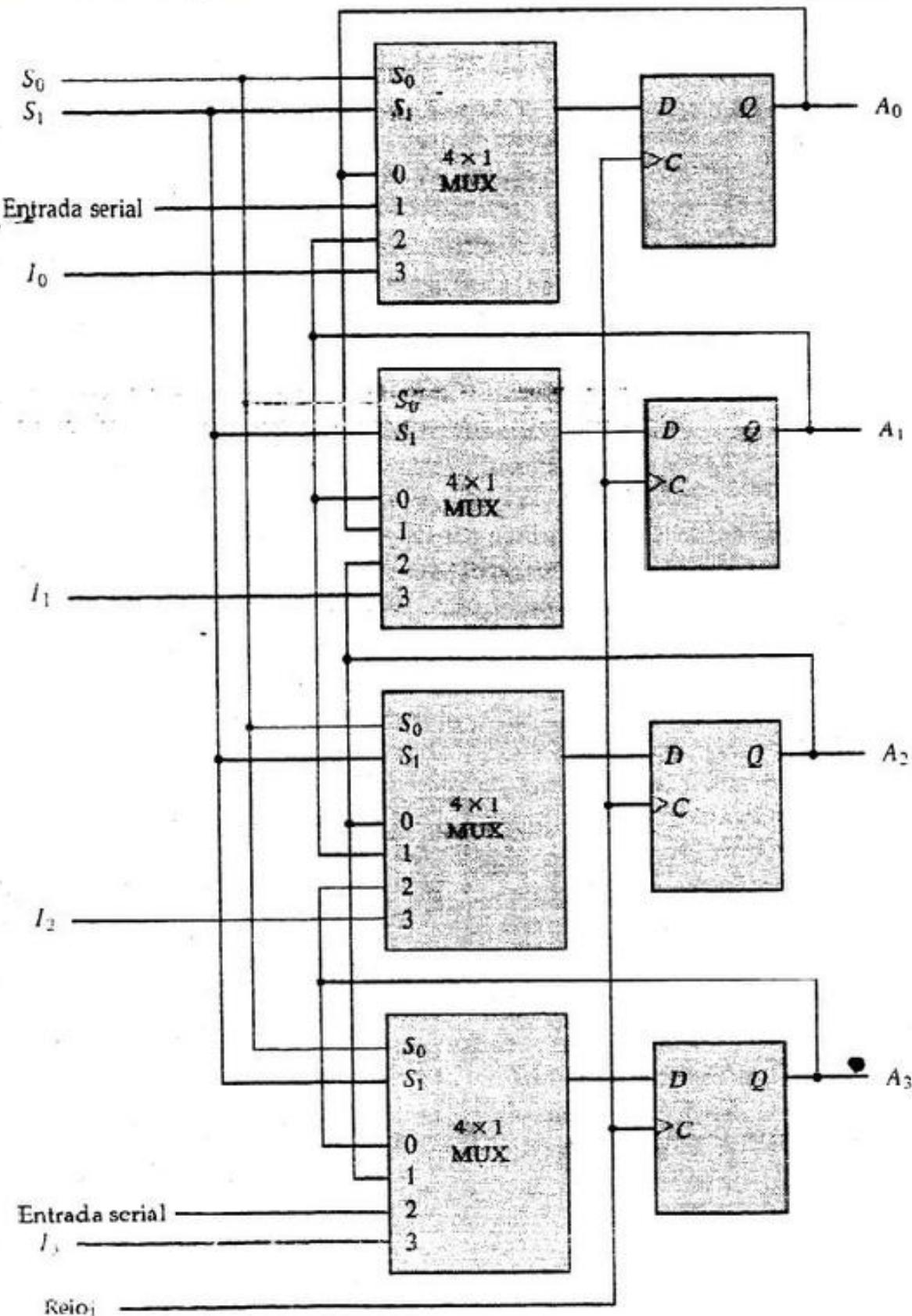


Figura 2-9 Registro de desplazamiento bidireccional con carga paralela.

costoso usar n líneas para transmitir los n bits en paralelo. Puede ser más económico utilizar una sola línea y transmitir la información en forma serial un bit cada vez. El transmisor carga el dato de n bits en paralelo en un registro de corrimiento y transmite entonces el dato desde la línea de salida serial. El receptor acepta el dato en forma serial en el registro de desplazamiento a través de su línea de entrada serial. Cuando la totalidad de los n

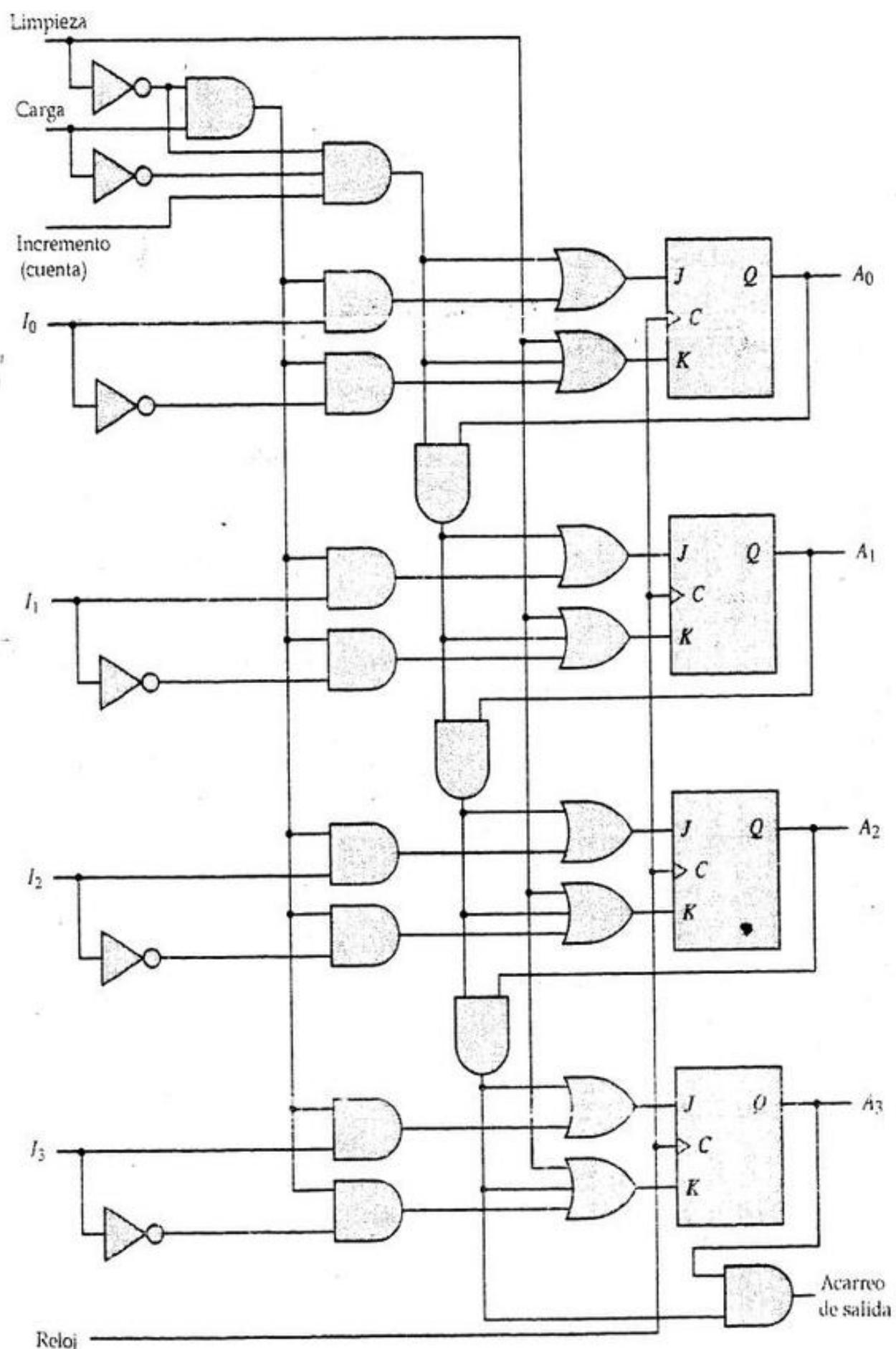


Figura 2-11 Contador binario de 4 bits con carga paralela y limpieza síncrona.

CAPÍTULO TRES

Representación de datos

EN ESTE CAPÍTULO

- 3-1 Tipos de datos
- 3-2 Complementos
- 3-3 Representación con punto fijo
- 3-4 Representación con punto flotante
- 3-5 Otros códigos binarios
- 3-6 Códigos de detección de error

3-1 Tipos de datos

En las computadoras digitales la información binaria se almacena en la memoria o en los registros del procesador. Los registros contienen datos o información de control. La información de control es un bit o un grupo de bits que se utilizan para especificar la secuencia de señales de comando necesarias para manipular los datos en otros registros. Los datos son números y otra información en código binario sobre los que se realizan operaciones para conseguir los resultados computacionales requeridos. En este capítulo presentamos los tipos de datos más comunes que se encuentran en las computadoras digitales y mostramos cómo los diversos tipos de datos se representan en forma de código binario en los registros de computadoras.

Los tipos de datos que se encuentran en los registros de las computadoras digitales pueden clasificarse en algunas de las siguientes categorías: 1) números que se utilizan en cálculos aritméticos; 2) letras del alfabeto que se utilizan en el procesamiento de datos, y 3) otros símbolos discretos que se utilizan con propósitos específicos. Todos los tipos de datos, excepto los números binarios, se representan en los registros de la computadora en forma de código binario. Esto es porque los registros están formados de

CAPÍTULO CUATRO

Transferencia de registro y microoperaciones

EN ESTE CAPÍTULO

- 4-1 Lenguaje de transferencia de registros
- 4-2 Transferencia de registros
- 4-3 Transferencias de bus y de memoria
- 4-4 Microoperaciones aritméticas
- 4-5 Microoperaciones lógicas
- 4-6 Microoperaciones de corrimiento
- 4-7 Unidad de corrimiento lógico aritmético

4-1 Lenguaje de transferencia de registros

Un sistema digital es una interconexión de módulos de hardware digital que realizan una tarea específica de procesamiento de información. Los sistemas digitales varían en tamaño y complejidad desde unos cuantos circuitos integrados hasta un complejo de computadoras digitales interconectadas e interactivas. El diseño de sistemas digitales utiliza de manera invariable un enfoque modular. Los módulos se construyen a partir de componentes digitales como registros, decodificadores, elementos aritméticos y lógica de control. Los diferentes módulos están interconectados con los datos y las trayectorias de control comunes para formar un sistema de computadora digital.

Los módulos digitales se definen mejor por los registros que contienen y las operaciones que realizan sobre los datos que almacenan. Las operaciones que se ejecutan sobre los datos almacenados en los registros se llaman microoperaciones. Una microoperación es una operación básica realizada sobre la información almacenada en uno o más registros. El resultado de la operación puede sustituir la información binaria anterior de un registro o puede transferirse a otro. Algunos ejemplos de microoperaciones son despla-

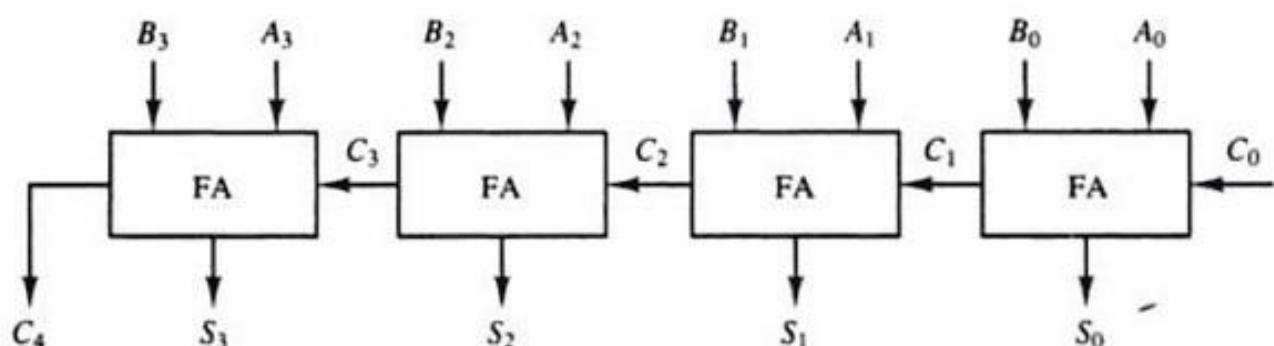


Figura 4-6 Sumador binario de 4 bits.

datos para las entradas A provienen de un registro (por ejemplo $R1$), y los n bits de datos para las B entradas provienen de otro registro (por ejemplo $R2$). La suma puede transferirse a un tercer registro o a alguno de los registros fuente ($R1$ o $R2$), sustituyendo su contenido previo.

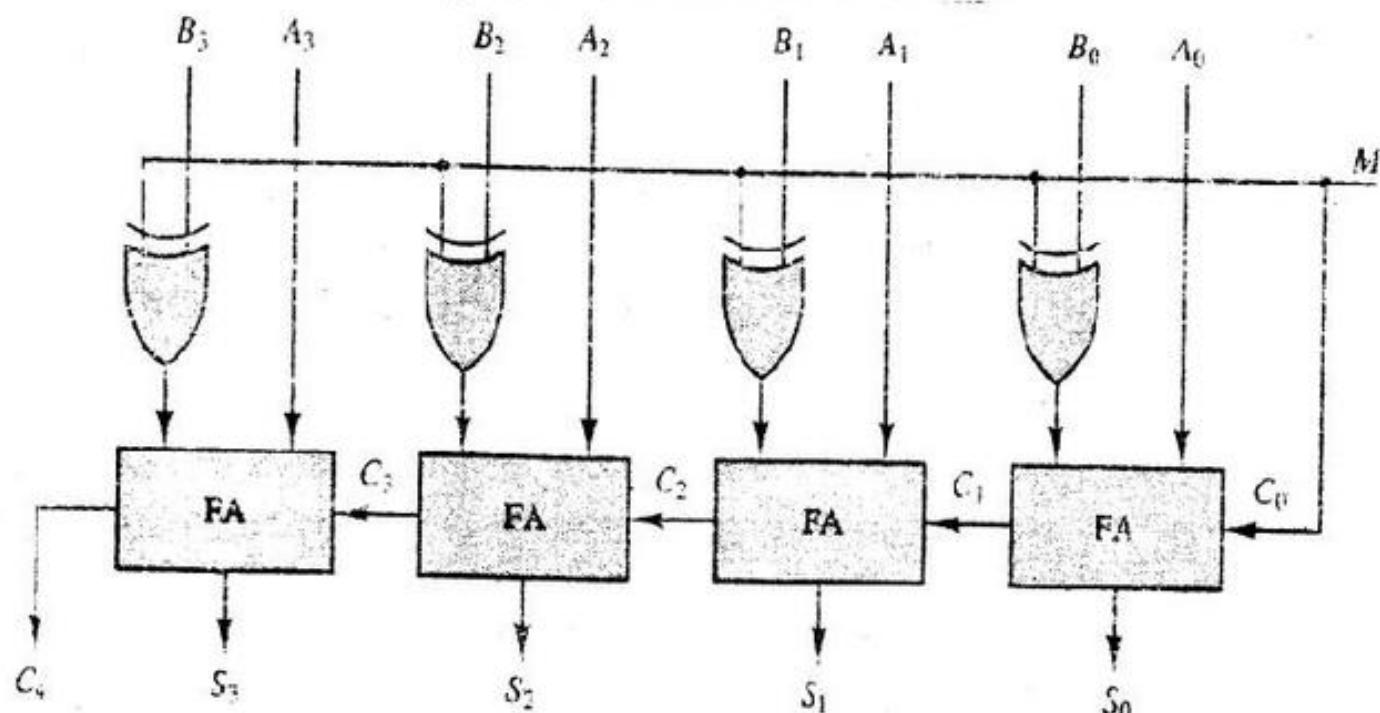
Sumador sustractor binario

La resta de números binarios puede realizarse en forma más conveniente por medio de complementos, según se analizó en la sección 3-2. Recuerde que la resta $A - B$ puede realizarse al tomar el complemento a 2 de B y sumarlo a A . El complemento a 2 puede obtenerse al tomar el complemento a 1 y agregar 1 al par de bits menos significativo. El complemento a 1 se puede obtener con inversores y se puede agregar 1 a la suma mediante el acarreo de entrada.

Las operaciones de suma y resta se pueden combinar en un circuito común al incluir una compuerta OR exclusiva con cada sumador completo. Un circuito sumador sustractor de 4 bits se muestra en la figura 4-7. La entrada de modo M controla la operación. Cuando $M = 0$ el circuito es un

sumador sustractor

Figura 4-7 Sumador sustractor de 4 bits.



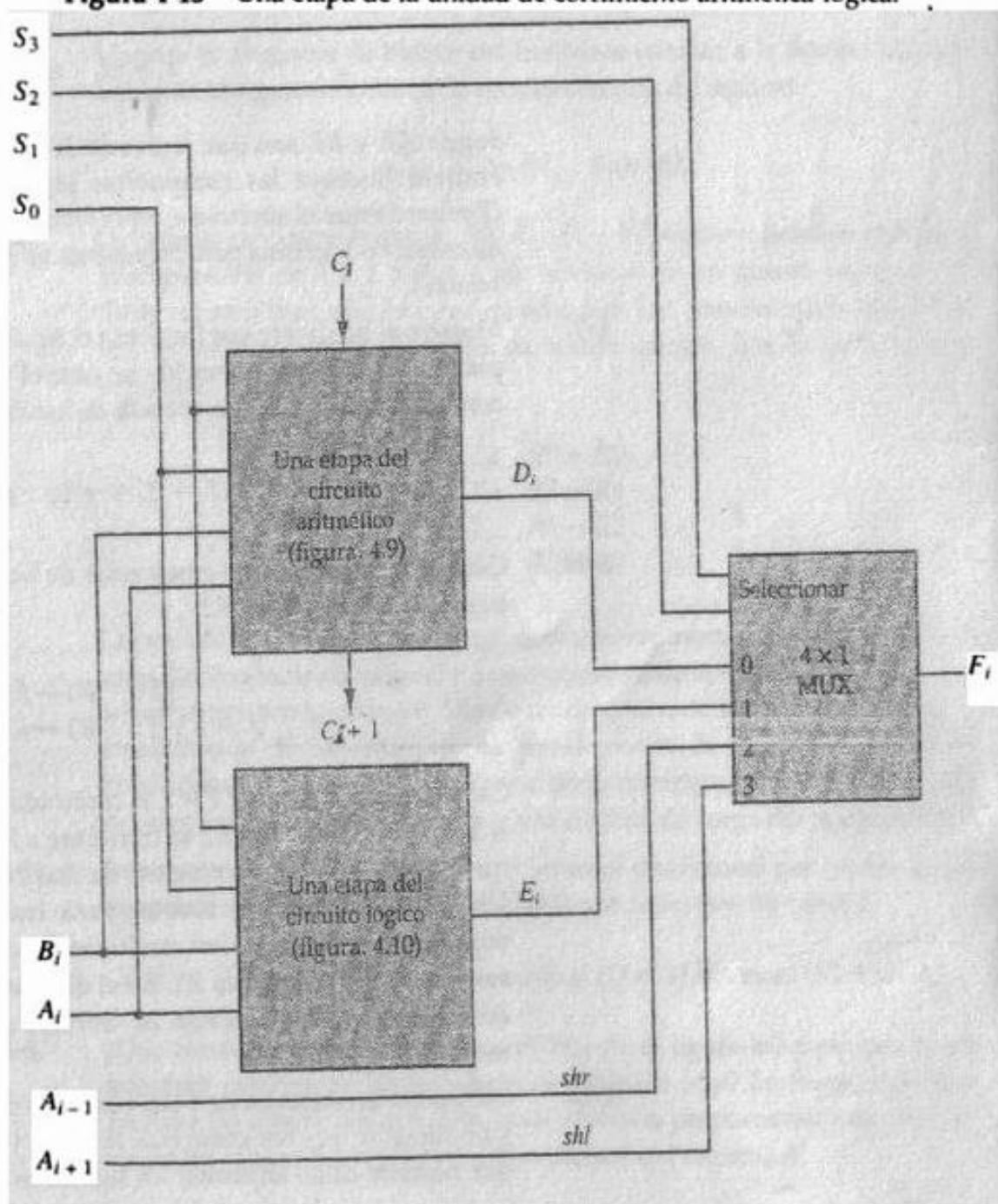
Cuando $S = 1$, los datos de entrada se desplazan a la izquierda (arriba en el diagrama). La tabla de función en la figura 4-12 muestra cuál entrada va a cada salida después del corrimiento. Un circuito por corrimiento con n entradas de datos y salidas necesita n multiplexores. Las dos entradas seriales pueden controlarse mediante otro multiplexor para proporcionar los tres tipos de desplazamiento posibles.

4-7 Unidad de corrimiento lógico aritmético

ALU

En lugar de tener registros individuales ejecutando las microoperaciones en forma directa, los sistemas computacionales emplean cierta cantidad de registros de almacenamiento conectados a todas las unidades operacionales comunes, la cual se denomina unidad aritmética-lógica, y se abrevia ALU

Figura 4-13 Una etapa de la unidad de corrimiento aritmética-lógica.



CAPÍTULO CINCO

Organización y diseño básico de computadoras

EN ESTE CAPÍTULO

- 5-1 Códigos de instrucción
- 5-2 Registros de computadora
- 5-3 Instrucciones de computadora
- 5-4 Temporización y control
- 5-5 Ciclo de instrucción
- 5-6 Instrucciones de referencia a memoria
- 5-7 Entrada-salida e interrupción
- 5-8 Descripción completa de una computadora
- 5-9 Diseño de una computadora básica
- 5-10 Diseño de un acumulador lógico

5-1 Códigos de instrucción

En este capítulo presentamos una computadora básica y mostramos cómo puede especificarse su operación con enunciados de transferencia de registros. La organización de la computadora se define mediante sus registros internos, la estructura de temporización y control, y el conjunto de instrucciones que utiliza. Después se lleva a cabo, en detalle, el diseño de la computadora. Aunque la computadora básica que se presenta en este capítulo es muy pequeña en comparación con las computadoras comerciales, tiene la ventaja de ser lo suficientemente simple para mostrar el proceso de diseño sin demasiadas complicaciones.

La organización interna de un sistema digital está definida por la secuencia de microoperaciones que ejecuta sobre los datos almacenados en sus registros. La computadora digital de propósito general puede ejecutar varias microoperaciones y, además, puede recibir instrucciones acerca de la secuencia específica de operaciones que debe realizar. El usuario de una computadora puede controlar el proceso mediante un programa. Un programa es un conjunto de instrucciones que especifican las operaciones, operandos y la secuencia mediante la cual tiene que ocurrir el procesamiento.

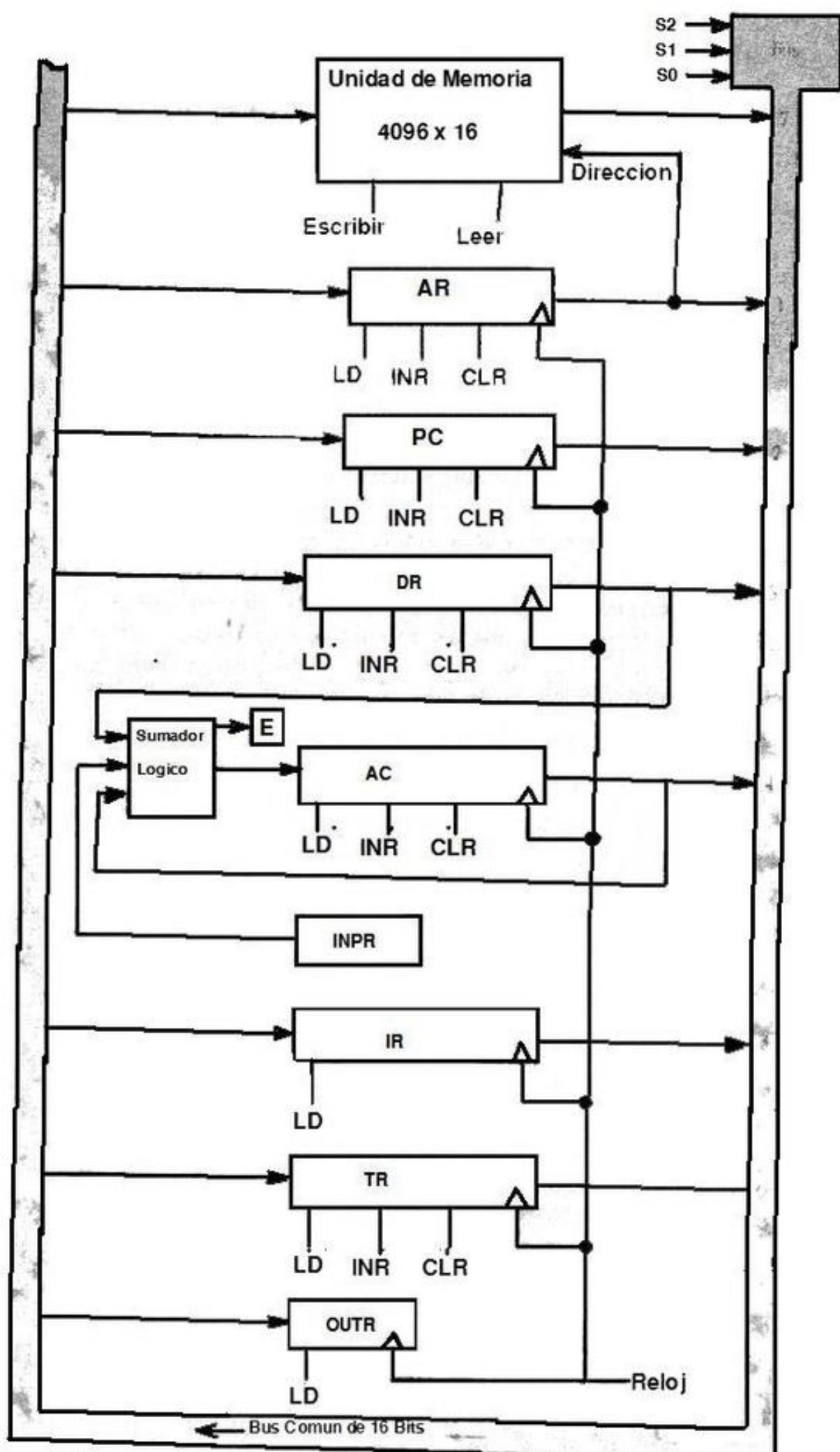


Figura 5-4 Registros de la Computadora básica conectados a un bus común.

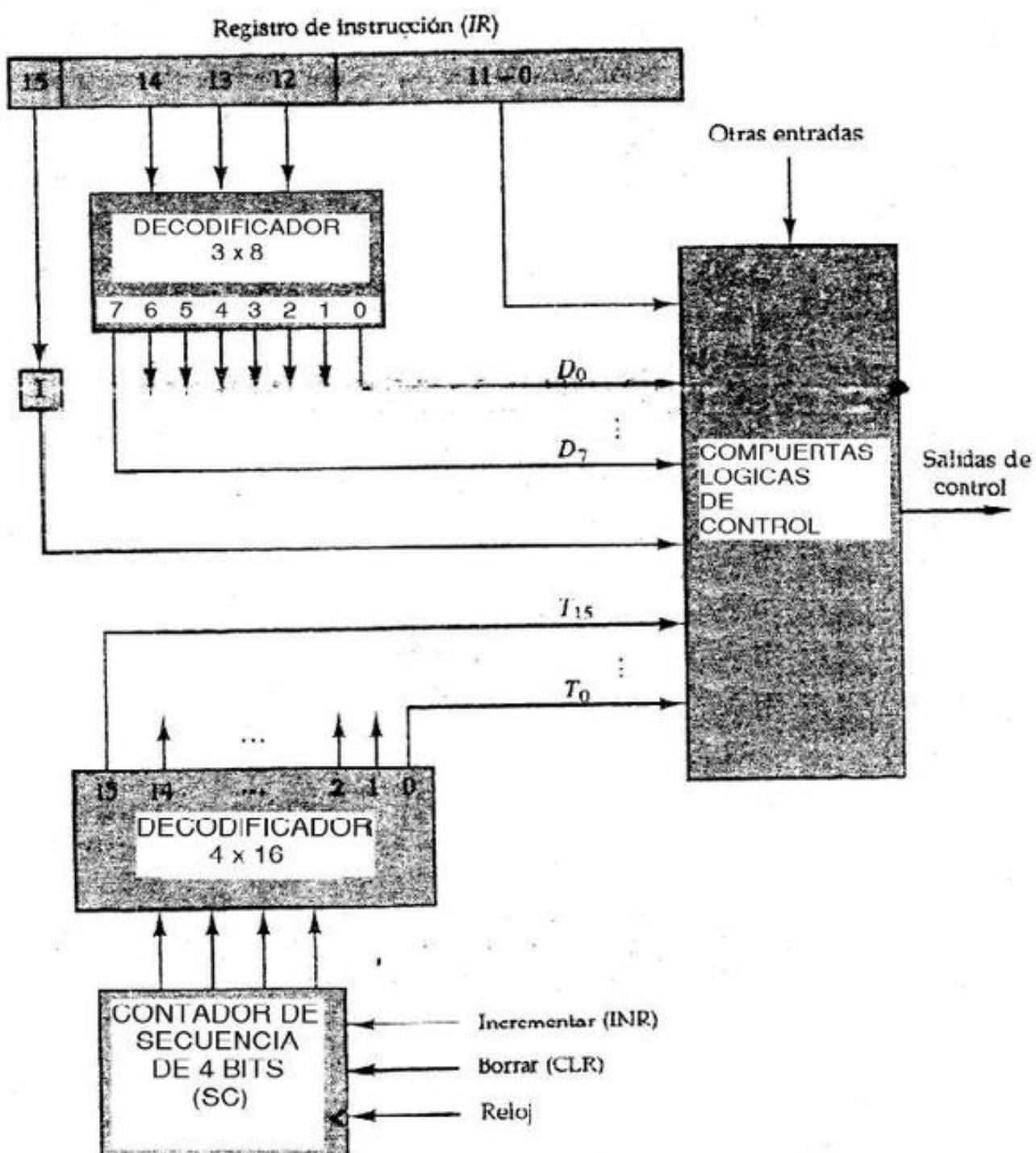


Figura 5-6 Unidad de control de la computadora básica.

La transición de reloj positiva T_0 en el diagrama activará solamente aquellos registros cuyas entradas de control están conectadas a la señal de temporización T_0 . SC se incrementa con cada transición de reloj positiva, a menos que su entrada CLR esté activa. Esto produce las secuencias y señales de temporización T_0, T_1, T_2, T_3, T_4 y así sucesivamente, según se muestra en el diagrama. (Nótese la relación entre la señal de temporización y la transición de reloj positiva correspondiente.) Si SC no se borra, las señales de temporización continuarán con T_5, T_6 hasta T_{15} y de regreso a T_0 .

Las últimas tres ondas de la figura 5-7 muestran cómo se borra SC cuando $D_3T_4 = 1$. La salida D_3 del decodificador de operación se activa al

EN ESTE CAPÍTULO

- 6-1** Introducción
- 6-2** Lenguaje de máquina
- 6-3** Lenguaje ensamblador
- 6-4** El ensamblador
- 6-5** Ciclos del programa
- 6-6** Programación de operaciones aritméticas y lógicas
- 6-7** Subrutinas
- 6-8** Programación de entrada-salida

6-1 Introducción

Un sistema de computadora total incluye tanto circuitería (*hardware*) como *programación* (*software*). El hardware consta de los componentes físicos y todo el equipo asociado. El software se refiere a los programas que están escritos para la computadora. Es posible conocer diferentes aspectos de programación de computadora sin relacionarse con los detalles de cómo opera la circuitería. También es posible diseñar partes de la circuitería sin conocer las posibilidades de su software. Sin embargo, quienes se interesan en la arquitectura de la computadora deben conocer tanto la circuitería como el software, porque los dos aspectos influyen uno en el otro.

Escribir un programa para una computadora consiste en especificar, en forma directa o indirecta, una secuencia de instrucciones de máquina. Las instrucciones de máquina dentro de la computadora forman un patrón binario con el que a los usuarios les resulta difícil trabajar y entender, si no es que imposible. Es preferible escribir programas con los símbolos más familiares del conjunto de caracteres alfanuméricos. Como consecuencia, existe la necesidad de traducir los programas simbólicos orientados al usuario, a programas binarios que reconozca el hardware.

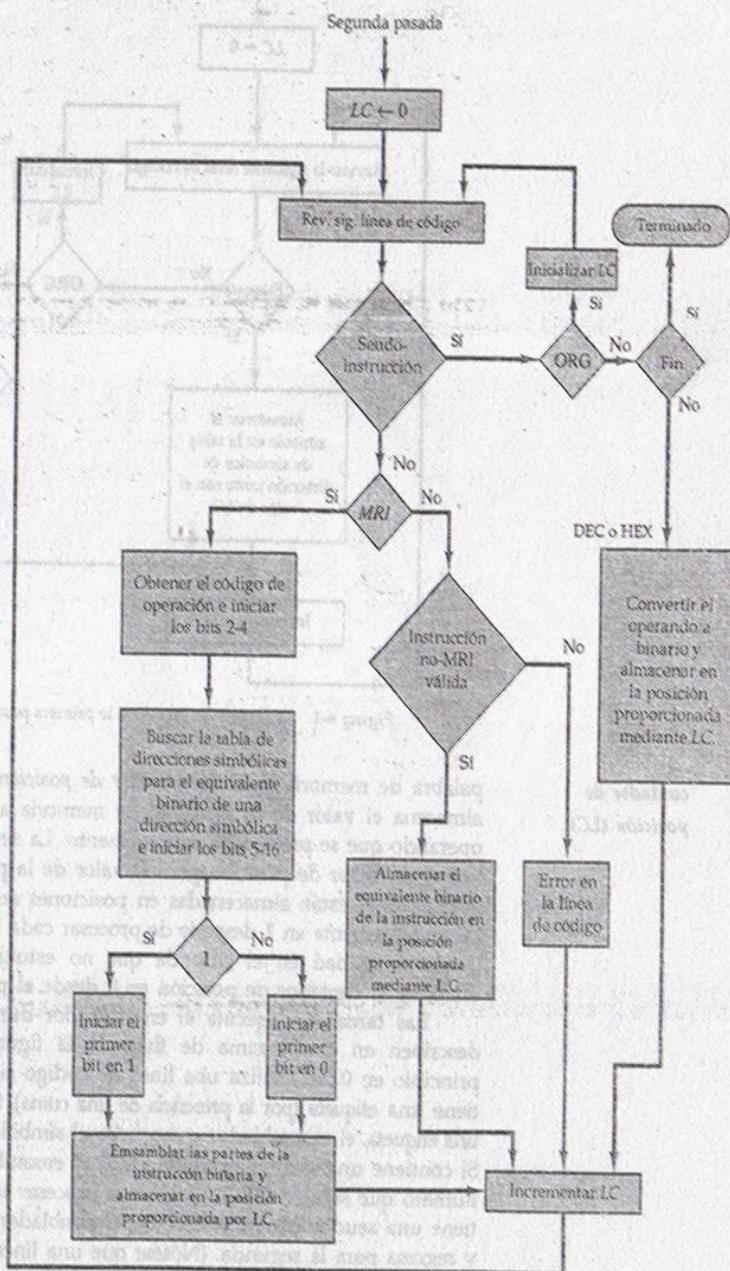


Figura 6-2 Diagrama de flujo para la segunda pasada del ensamblador.

EN ESTE CAPÍTULO

- 7-1 Memoria de control
- 7-2 Secuencia de la dirección
- 7-3 Ejemplo de un microprograma
- 7-4 Diseño de la unidad de control

7-1 Memoria de control

La función de la unidad de control en una computadora digital es iniciar secuencias de microoperaciones. La cantidad de tipos de operaciones diferentes que están disponibles en un sistema dado es finita. La complejidad del sistema digital se deriva de la cantidad de secuencias de microoperaciones que se ejecutan. Cuando la circuitería genera señales de control por medio de técnicas de diseño lógico convencional, se dice que la unidad de control es por *circuitería*. La microprogramación es una segunda alternativa para diseñar la unidad de control de una computadora digital. El principio de microprogramación es un método elegante y sistemático para controlar las secuencias de microoperaciones en una computadora digital.

La función de control que especifica una microoperación es una variable binaria. Cuando se halla en un estado binario se ejecuta la microoperación correspondiente. Una variable de control en estado binario opuesto no cambia el estado de los registros en el sistema. El estado activo de la variable de control puede ser el estado 1 o el estado 0, según la aplicación. En un sistema organizado con bus, las señales de control que especifican microoperaciones son grupos de bits que seleccionan las trayectorias en los multiplexores, decodificadores y unidades lógicas aritméticas.

La unidad de control inicia una serie de pasos secuenciales de microoperaciones. Durante cualquier momento, se van a iniciar ciertas microoperaciones, mientras otras quedan inactivas. Las variables de control pueden

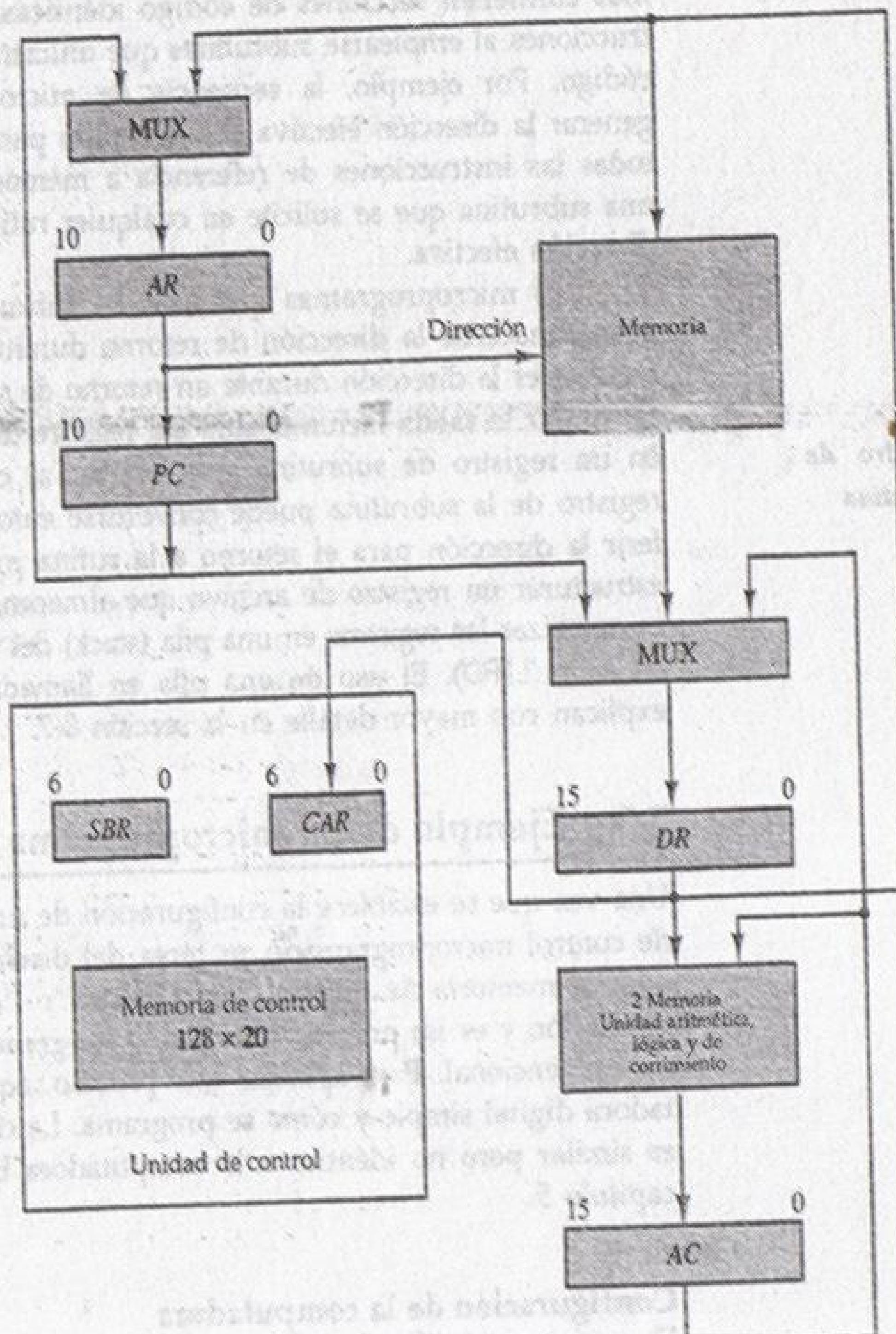


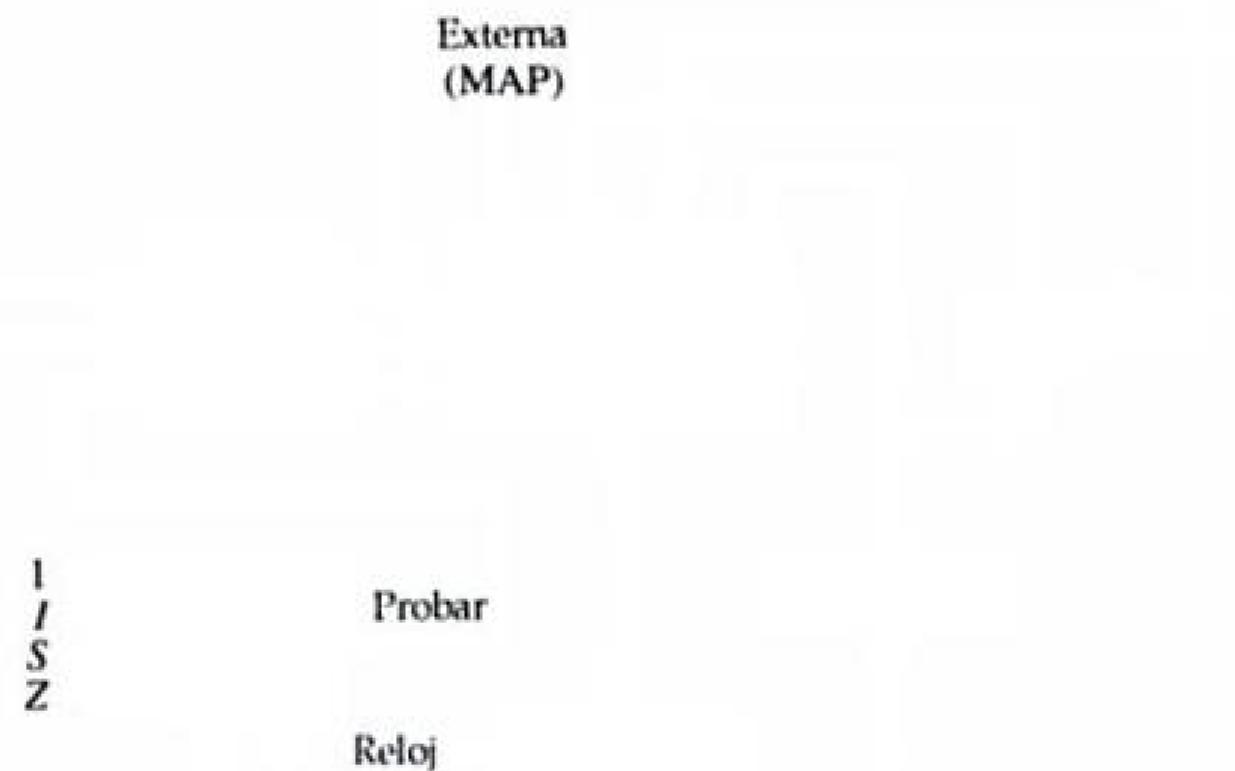
Figura 7-4 Configuración de la circuitería de computadora.

La transferencia de información entre los registros del procesador se realiza mediante multiplexores, no por el bus común. *DR* puede recibir información de *AC*, *PC* o de la memoria. *AR* puede recibir información de *PC* o *DR*. *PC* sólo puede recibir información de *AR*. La unidad aritmética, lógica y de corrimiento realiza microoperaciones con datos de *AC* y *DR* y coloca el resultado en *AC*. Nótese que la memoria recibe su dirección de *AR*. Los datos de entrada escritos en la memoria provienen de *DR*, y los datos que se leen en la memoria sólo pueden ir a *DR*.

El formato de instrucción de computadora aparece en la figura 7-5(a). Consta de tres campos: un campo de 1 bit para direccionamiento indirecto

Para mostrar la estructura interna de un secuenciador de microprograma típico analizaremos una unidad particular que es conveniente para usarse en el ejemplo de computadora de microprograma desarrollado en la sección anterior. El diagrama de bloque del secuenciador de microprograma se muestra en la figura 7-8. La memoria de control se incluye en el diagrama para mostrar la interacción entre el secuenciador y la memoria conectada a él. Hay dos multiplexores en el circuito. El primero selecciona una dirección de las cuatro fuentes y le marca una ruta hacia dentro del registro de dirección de control *CAR*. El segundo multiplexor prueba el valor de un bit de estado seleccionado y el resultado de la prueba se aplica a un circuito lógico de entrada. La salida de *CAR* proporciona la dirección para la memoria de control. El contenido de *CAR* se incrementa y se aplica a una de las entradas del multiplexor y al registro de subrutina *SBR*. Las otras tres entradas al multiplexor número 1 provienen del campo de dirección de la microinstrucción presente, de la salida de *SBR* y de una fuente externa que

Figura 7-8 Secuenciador de microprograma para una memoria de control.



EN ESTE CAPÍTULO

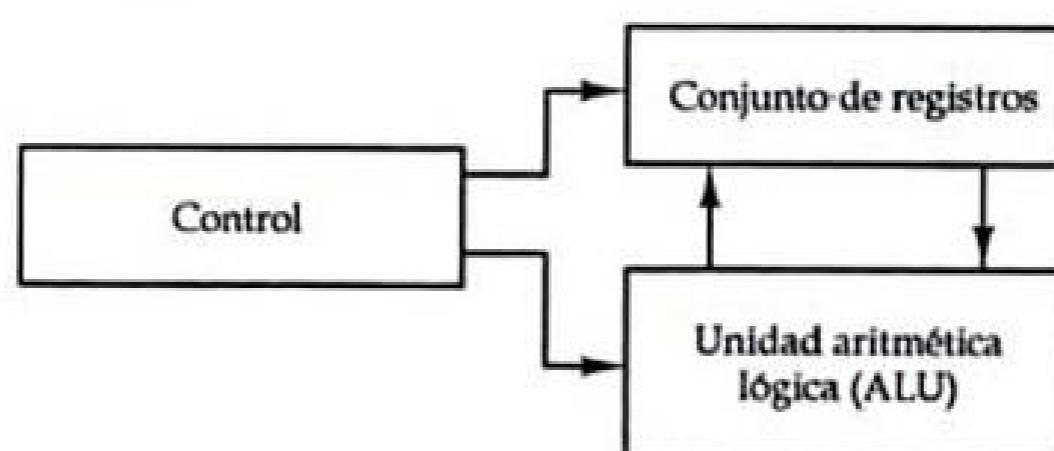
- 8-1 Introducción
- 8-2 Organización general de los registros
- 8-3 Organización de una pila
- 8-4 Formatos de las instrucciones
- 8-5 Modos de direccionamiento
- 8-6 Transferencia y manipulación de los datos
- 8-7 Control del programa
- 8-8 Computadora de conjunto de instrucciones reducido (RISC)

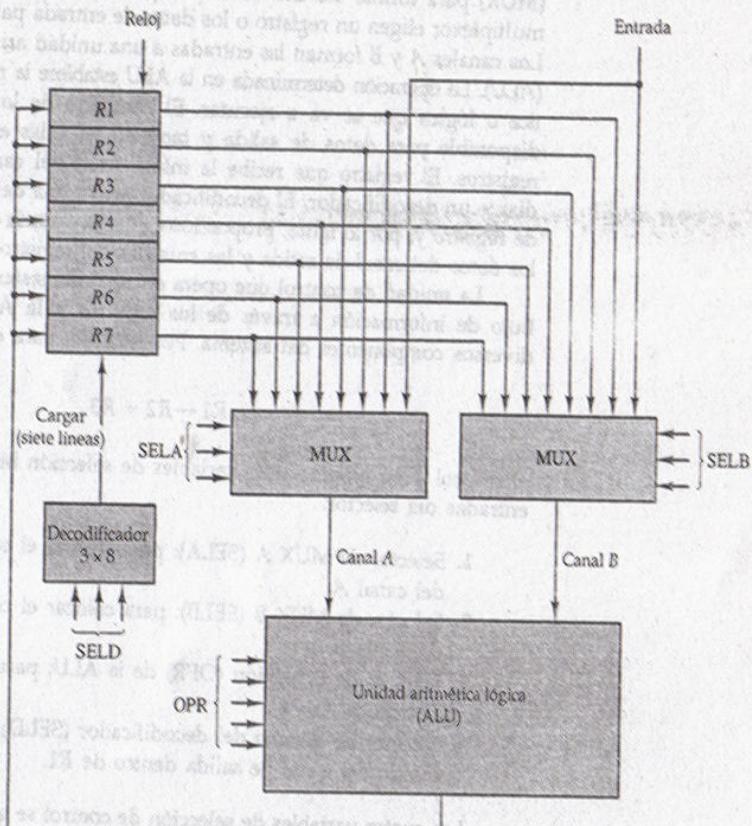
8-1 Introducción

CPU

La parte de la computadora que ejecuta el grueso de las operaciones de procesamiento de datos se llama unidad de procesamiento central y se denomina CPU. La CPU está formada de tres partes principales, como se muestra en la figura 8-1. El conjunto de registros almacena datos intermedios que se usan durante la ejecución de las instrucciones. La unidad aritmética-lógica (ALU) lleva a cabo las microoperaciones requeridas para ejecutar las instrucciones. La unidad de control supervisa la transferencia de información entre los registros e instruye a la ALU sobre cuál operación ejecutar.

Figura 8-1 Componentes principales de una CPU.





a) Diagrama de bloque

3	3	3	5
SEL A	SEL B	SEL D	OPR

b) Palabra de control

Figura 8-2 Conjunto de registros con ALU común.



Imagen protegida por derechos de autor

3000

3997

3998

3999

4000

4001

DR

Figura 8-4 Memoria de computadora con segmentos de programa, datos y pila.

$$SP \leftarrow SP - 1$$

$$M[SP] \leftarrow DR$$

El apuntador de pila se decrementa para que apunte en la dirección de la siguiente palabra. Una operación de escritura en la memoria inserta la palabra de *DR* dentro de la parte superior de la pila. Se recupera un nuevo dato con una instrucción push, de la manera siguiente:

$$DR \leftarrow M[SP]$$

$$SP \leftarrow SP + 1$$

instrucciones push y pop desde una pila de memoria. Sin embargo, el procesador no tiene instrucciones de tipo de direccionamiento cero, las cuales son características de una CPU organizada con pila.

Para dar una ejemplo de la influencia de la cantidad de direcciones sobre los programas de computadora, evaluaremos el siguiente enunciado aritmético

$$X = (A + B) * (C + D)$$

utilizando cero, uno, dos o tres instrucciones de direccionamiento. Emplearemos los símbolos ADD, SUB, MUL y DIV para las cuatro operaciones aritméticas; MOV para la operación de tipo transferencia; y LOAD y STORE para transferencias hacia y desde la memoria y el registro AC. Consideraremos que los operandos en las direcciones de memoria A, B, C y D y el resultado deben almacenarse en la memoria en la dirección X.

Instrucciones de tres direcciones

Las computadoras con formato de instrucción de tres direcciones pueden utilizar cada campo de dirección para especificar un registro de procesador o un operando de memoria. En seguida, se muestra el programa en lenguaje ensamblador que evalúa $X = (A + B) * (C + D)$, junto con comentarios que explican la operación de transferencia de registros de cada instrucción.

ADD	R1, A, B	$R1 \leftarrow M[A] + M[B]$
ADD	R2, C, D	$R2 \leftarrow M[C] + M[D]$
MUL	X, R1, R2	$M[X] \leftarrow R1 * R2$

Se considera que la computadora tiene dos registros de procesador, R1 y R2. El símbolo $M[A]$ representa el operando en la dirección de memoria que se simboliza mediante A.

La ventaja del formato de tres direcciones es que produce a la brevedad programas cuando evalúa expresiones aritméticas. La desventaja es que las instrucciones de código binario requieren demasiados bits para especificar tres direcciones. Un ejemplo de una computadora comercial que utiliza instrucciones de tres direcciones es la Cyber 170. Los formatos de instrucciones en la computadora Cyber están limitados a tres campos de dirección de registro o a dos campos de dirección de registro y a un campo de dirección de memoria.

Instrucciones de dos direcciones

Las instrucciones de dos direcciones son las más comunes en las computadoras comerciales. En ellas, también cada campo de dirección puede especificar un registro de procesador o una palabra de memoria. El programa para evaluar $X = (A + B) * (C + D)$ es como sigue:

instrucción antes de que se haga la referencia real al operando. Las computadoras utilizan técnicas de modo de direccionamiento para acomodar una o las dos siguientes consideraciones:

1. Proporcionar al usuario versatilidad de programación al ofrecer facilidades como apuntadores a memoria, contadores para control de ciclo, indexación de datos y reubicación de programas.
2. Reducir la cantidad de bits en el campo de direccionamiento de la instrucción.

La disponibilidad de los modos de direccionamiento proporciona al programador con experiencia en lenguaje ensamblador la flexibilidad para escribir programas más eficientes en relación con la cantidad de instrucciones y el tiempo de ejecución.

Para comprender los diferentes modos de direccionamiento que se presentarán en esta sección, es imperativo entender el ciclo de operación básico de la computadora. La unidad de control de una computadora está diseñada para recorrer un ciclo de instrucciones que se divide en tres fases principales:

1. Búsqueda de la instrucción de la memoria.
2. Decodificar la instrucción.
3. Ejecutar la instrucción.

Hay un registro en la computadora llamado contador de programa o PC, que lleva un registro de las instrucciones del programa almacenado en la memoria. PC contiene la dirección de la siguiente instrucción que se va a ejecutar y se incrementa cada vez que se recupera una instrucción de la memoria. La decodificación realizada en el paso 2 determina la operación que se va a ejecutar, el modo de direccionamiento de la instrucción y la posición de los operandos. Después la computadora ejecuta la instrucción y regresa al paso 1 para hacer la búsqueda de la siguiente instrucción en secuencia.

En algunas computadoras el modo de direccionamiento de la instrucción se especifica con un código binario distinto, como se hace con el código de operación. Otras computadoras utilizan un código binario único que representa la operación y el modo de la instrucción. Pueden definirse instrucciones con diversos modos de direccionamiento y, en ocasiones, se combinan dos o más modos de direccionamiento en una instrucción.

Un ejemplo de un formato de instrucción con un campo de modo de direccionamiento distinto se muestra en la figura 8-6. El código de operación especifica la operación que se va a ejecutar. El campo de modo se utiliza para ubicar los operandos necesarios para la operación. Puede haber o no un campo de direccionamiento en la instrucción. Si hay uno, puede representar una dirección de memoria o un registro de procesador. Además, como se

analizó en la sección anterior, la instrucción puede tener más de un campo de dirección, y cada campo de dirección puede estar asociado con su propio modo de direccionamiento particular.

Aunque la mayoría de los modos de direccionamiento modifican el campo de dirección de la instrucción, hay dos modos que no necesitan el campo de dirección. Son los modos implícito e inmediato.

Modo implícito: En este modo se especifican los operandos en forma implícita en la definición de la instrucción. Por ejemplo, la instrucción "complementar acumulador" es una instrucción de modo implícito porque el operando en el registro de acumulador está implícito en la definición de la instrucción. De hecho todas las instrucciones de referencia a registro que utilizan un acumulador son instrucciones de modo implícito.

Código de operación	Modo	Dirección
---------------------	------	-----------

Figura 8-6. Formato de instrucciones con campo de modo.

Las instrucciones de dirección cero en una computadora organizada con pila son instrucciones de modo implícito porque está implícito que los operandos están en la parte superior de la pila.

Modo inmediato: En este modo se especifica el operando en la instrucción misma. En otras palabras, una instrucción de modo inmediato tiene un campo de operando, en lugar de un campo de dirección. Un campo de operando contiene el operando real que se va a usar junto con la operación especificada en la instrucción. Las instrucciones de modo inmediato son útiles para inicializar registros en un valor constante.

Se mencionó antes que el campo de dirección de una instrucción puede especificar una palabra de memoria o un registro de procesador. Cuando el campo de dirección especifica un registro de procesador se dice que la instrucción está en modo de registro.

Modo de registro: En este modo, los operandos están en registros que residen dentro de la CPU. Se selecciona el registro particular de un campo de registro en la instrucción. Un campo k bits puede especificar cualquiera de 2^k registros.

Modo indirecto por registro: En este modo la instrucción especifica un registro en la CPU cuyo contenido proporciona la dirección del operando en la memoria. En otras palabras, el registro seleccionado contiene la dirección del operando en lugar del operando mismo. Antes de utilizar una instrucción de modo indirecto por registro, el programador debe asegurarse

TABLA 8-4 Lista tabular de un ejemplo numérico

Modo de direccionamiento	Dirección efectiva	Contenido de AC
Direccionamiento directo	500	800
Operando inmediato	201	500
Direccionamiento indirecto	800	300
Direccionamiento relativo	702	325
Direccionamiento indexado	600	900
Registro	—	400
Indirecto por registro	400	700
Autoincremento	400	700
Autodecremento	399	450

8-6 Transferencia y manipulación de los datos

Las computadoras proporcionan un amplio conjunto de instrucciones para dar al usuario la flexibilidad de realizar diferentes tareas computacionales. El conjunto de instrucciones de diferentes computadoras varía de una a otra sobre todo en la manera en que los operandos se determinan de los campos de direcciones. Las operaciones reales disponibles en el conjunto de instrucciones no son muy diferentes de una computadora a otra. Lo que sucede es que las asignaciones de código binario y el campo de código de la operación son diferentes en diversas computadoras, aun para la misma operación. También puede suceder que el nombre simbólico asignado a instrucciones en la notación de lenguaje ensamblador sea diferente en varias computadoras, aun para la misma instrucción. No obstante, hay un conjunto básico de operaciones que gran parte, si no es que todas las computadoras incluyen en su repertorio de instrucciones. El conjunto de operaciones básicas disponibles en una computadora típica es el objeto que se cubre en esta sección y la siguiente.

conjunto de operaciones básicas

La mayor parte de instrucciones de computadora pueden clasificarse en tres categorías:

1. Instrucciones de transferencia de datos.
2. Instrucciones de manipulación de datos.
3. Instrucciones de control de programa.

Las instrucciones de transferencia de datos producen una transferencia de datos de una localidad a otra sin cambiar el contenido de la información binaria. Las instrucciones de manipulación de datos son las que realizan operaciones aritméticas, lógicas y de corrimiento. Las instrucciones de control de programa proporcionan posibilidades de toma de decisiones y cambian la trayectoria que toma un programa cuando se ejecuta en la computadora. El conjunto de instrucciones de una computadora particular determina

derecha junto con el resto del número, pero el bit de signo no cambia. Esta es una operación de corrimiento a la derecha en la que el bit final permanece igual. La instrucción de corrimiento aritmético a la izquierda inserta 0 en la posición final y es idéntica a la instrucción lógica de corrimiento a la izquierda. Por esta razón muchas computadoras no proporcionan una instrucción distinta de corrimiento aritmético a la izquierda cuando ya está disponible la instrucción de corrimiento lógico a la izquierda.

Las instrucciones de rotación producen un corrimiento circular. Los bits recorridos en un extremo de la palabra no se pierden en un desplazamiento lógico, pero se hacen circular hasta el otro extremo. La instrucción de rotación a través del bit de acarreo trata al bit de acarreo como una extensión del registro cuya palabra se está rotando. Por lo tanto, una instrucción de rotación a la izquierda a través del acarreo transfiere el bit de acarreo a la posición de bit a la extrema derecha del registro, transfiere la posición de la extrema izquierda al acarreo y, al mismo tiempo, recorre todo el registro a la izquierda.

Algunas computadoras tienen un formato de campo múltiple para las instrucciones de corrimiento. Un campo contiene el código de operación y los otros especifican el tipo de corrimiento y la cantidad de veces que se va a recorrer un operando. Un formato de código de instrucción posible de una instrucción de desplazamiento puede incluir cinco campos de la manera siguiente:

OP	REG	TYPE	RL	COUNT
----	-----	------	----	-------

Aquí OP es el campo de código de operación; REG es una dirección de registro que especifica la posición del operando; TYPE es un campo de 2 bits que especifica los cuatro diferentes tipos de corrimientos; LR es un campo de 1 bit que especifica un corrimiento a la derecha o a la izquierda y COUNT es un campo de k bits que especifica hasta $2^k - 1$ corrimientos. Con tal formato es posible especificar el tipo de corrimiento, la dirección y la cantidad de corrimientos, todo en una sola instrucción.

8-7 Control del programa

Las instrucciones siempre se almacenan en localidades de memoria sucesivas. Cuando se procesan en la CPU, las instrucciones se recuperan de localidades de memoria consecutivas y se ejecutan. Cada vez que se recupera una instrucción de la memoria, el contador de programa se incrementa para que contenga la dirección de la siguiente instrucción en secuencia. Después de la ejecución de una instrucción de transferencia de datos o de manipulación de datos, el control retorna al ciclo de recuperación con el contador de programa que contiene la dirección de la siguiente instrucción de secuencia. Por otra parte, un tipo de instrucción de control de programa,

Instrucciones de brinco condicional

La tabla 8-11 proporciona una lista de las instrucciones de transferencia del control del programa más comunes. Cada nemónico está formulado con la letra B (por branch, salto) y una abreviatura del nombre de la condición. Cuando se usa el estado de condición opuesta, se inserta la letra N (no) para definir el estado 0. Por lo tanto BC es brincar en el acarreo y BNC es brinar cuando no hay acarreo. Si la condición plantecada es cierta, se transfiere el control del programa a la dirección especificada por la instrucción. Si no, el control continúa con la instrucción que sigue. Las instrucciones condicionales pueden asociarse también con las instrucciones del tipo de control de programa brinar, interrumpe, llamar o retornar.

El bit de estado cero se utiliza para probar si el resultado de una operación en la ALU es igual a cero o no. El bit de acarreo se utiliza para comprobar si hay un acarreo de la posición de bit más significativa de la ALU. También se usa junto con las instrucciones de rotación para verificar el bit más significativo recorrido de un registro dentro de la posición de

TABLA 8-11 Instrucciones de transferencia de control condicional

Nemónico	Condición de transferencia	Condición probada
BZ	Brinar si hay cero	$Z = 1$
BNZ	Brinar si no hay cero	$Z = 0$
BC	Brinar si hay acarreo	$C = 1$
BNC	Brinar si no hay acarreo	$C = 0$
BP	Brinar si hay un signo más	$S = 0$
BM	Brinar si hay un signo menos	$S = 1$
BV	Brinar si hay sobreflujo	$V = 1$
BNV	Brinar si no hay sobreflujo	$V = 0$

Condiciones para comparar sin signo ($A - B$)

BHI	Brinar si es mayor	$A > B$
BHE	Brinar si es mayor o igual	$A \geq B$
BLO	Brinar si es menor	$A < B$
BLOE	Brinar si es menor o igual	$A \leq B$
BE	Brinar si es igual	$A = B$
BNE	Brinar si es diferente	$A \neq B$

Condiciones para comparar con signo ($A - B$)

BGT	Brinar si es mayor que	$A > B$
BGE	Brinar si es mayor o igual	$A \geq B$
BLT	Brinar si es menor que	$A < B$
BLE	Brinar si es menor o igual	$A \leq B$
BE	Brinar si es igual	$A = B$
BNE	Brinar si es diferente	$A \neq B$

acarreo. El bit de signo refleja el estado del bit más significativo de la salida de la ALU. $S = 0$ representa un signo positivo y $S = 1$ un signo negativo. Por lo tanto, un brinco sobre más comprueba un bit de signo en 0 y un brinco sobre menos comprueba un bit de signo en 1. Sin embargo, debe recordarse que estas dos instrucciones de brinco condicional pueden utilizarse para verificar el valor del bit más significativo ya sea que represente un signo o no. El bit de sobreflugo se utiliza junto con operaciones aritméticas realizadas sobre números en representación de complemento a 2 con signo.

Como se dijo antes, la instrucción de comparar ejecuta una resta de dos operandos, digamos $A - B$. El resultado de la operación no se transfiere a un registro destino, pero se afectan los bits de estado. El registro de estado proporciona información acerca de la magnitud relativa de A y B . Algunas computadoras proporcionan instrucciones de transferencia del control del programa condicionales que pueden aplicarse exactamente después de la ejecución de una instrucción de comparar. Las condiciones específicas que se van a probar dependen de si los dos números A y B se van a considerar con signo o sin signo. La tabla 8-11 proporciona una lista de tales instrucciones de transferencia condicional. Nótese que utilizamos las palabras más alta y más baja para denotar las relaciones entre números sin signo y mayor y menor que para números con signo. La magnitud relativa que se muestra bajo la columna de condición probada, en la tabla, parece igual para los números con signo y sin signo. Sin embargo, este no es el caso porque cada uno debe considerarse en forma separada, como se explica en el siguiente ejemplo numérico.

ejemplo numérico

Consideremos una ALU de 8 bits, como se muestra en la figura 8-8. El número sin signo más grande que puede acomodarse en 8 bits es 255. El rango de números con signo está entre +127 y -128. La resta de dos números es igual, ya sea que no tengan signo o estén en representación de complemento a 2 con signo (véase el capítulo 3). Sean $A = 11110000$ y $B = 00010100$. Para ejecutar $A - B$, la ALU toma el complemento a 2 de B y lo suma a A .

$$\begin{array}{r} A: 11110000 \\ \overline{B} + 1: +11101100 \\ \hline A - B: 11011100 \end{array} \quad C = 1 \quad S = 1 \quad V = 0 \quad Z = 0$$

La instrucción de comparar actualiza el bit de estado según se muestra. $C = 1$ porque hay un acarreo de la última etapa. $S = 1$ porque el bit a la extrema izquierda es 1. $V = 0$ porque los últimos dos acarreos son iguales a 1 y $Z = 0$ porque el resultado es diferente de 0.

Si consideramos números sin signo, el equivalente decimal de A es 240 y el de B es 20. La resta en decimal es $240 - 20 = 220$. Por lo tanto, el resultado binario 11011100 es el equivalente del decimal 220. Entonces $240 > 20$, tenemos que $A > B$ y $A \neq B$. Estas dos relaciones también pueden derivarse de que el bit de estado C es igual a 1 y el bit Z es igual a 0. Las instrucciones que producirán una transferencia de control después de esta comparación

instrucción al realizar dos operaciones: 1) se almacena la dirección de la siguiente instrucción disponible en el contador de programa (la dirección de retorno en una posición temporal, para que la subrutina sepa a dónde retornar) y 2) se transfiere el control al comienzo de la subrutina. La última instrucción de cada subrutina, por lo general llamada *retorno de la subrutina*, transfiere la dirección de retorno de la posición temporal dentro del contador de programa. Esto da como resultado una transferencia del control del programa a la instrucción cuya dirección se almacenaba originalmente en la posición temporal.

Diversas computadoras utilizan una posición temporal diferente para almacenar la dirección de retorno. Algunas la almacenan en la primera posición de memoria de subrutina, otras en una posición fija en la memoria, algunas más en un registro de procesador y otras en una pila de memoria. La manera más eficiente es almacenar la dirección de retorno en una pila de memoria. La ventaja de utilizar una pila para la dirección de retorno es que cuando se solicitan subrutinas sucesivas, las direcciones de retorno secuenciales pueden salvase dentro de la pila. El retorno de la instrucción de subrutina lee (pop) la pila y el contenido de su parte superior se transfiere al contador de programa. De esta manera, el retorno es siempre al programa que solicitó la última vez una subrutina. Una llamada de subrutina se implanta con las siguientes microoperaciones:

$$SP \leftarrow SP - 1$$

Decrementar apuntador de pila

$$M[SP] \leftarrow PC$$

Salvar el contenido de PC sobre la pila

$$PC \leftarrow$$

Transferir control a subrutina

Si se solicita otra subrutina mediante la actual, la nueva dirección de retorno se salva dentro de la pila, y así sucesivamente. La instrucción que hace retornar de la última subrutina se realiza mediante las microoperaciones:

$$PC \leftarrow M[SP] \quad \text{Recuperar la pila y transferir a PC}$$

$$SP \leftarrow SP + 1 \quad \text{Incrementar apuntador de pila}$$

Al utilizar una pila de subrutina, la circuitería almacena en forma automática todas las direcciones de retorno en una unidad. El programador no tiene que preocuparse o recordar dónde se almacenó la dirección de retorno.

Una *subrutina recursiva* es una subrutina que se solicita a sí misma. Si sólo se utiliza un registro o localidad de memoria para almacenar la dirección de retorno y la subrutina recursiva se solicita a sí misma, destruye la

dirección de retorno anterior. Esto no es deseable porque se destruye información vital. Este problema puede resolverse si se emplean diferentes localidades de almacenamiento para cada utilización de subrutina mientras otra utilización de nivel más ligero está activa todavía. Cuando se utiliza una pila, cada dirección de retorno puede salvarse dentro de la pila sin destruir ningún valor previo. Esto resuelve el problema de la subrutina recursiva porque la siguiente subrutina que sale es siempre la última subrutina que se solicitó.

Interrupción del programa

El concepto de interrupción de programa se utiliza para manejar diversos problemas que surgen de la secuencia de programa normal. La interrupción de programa se refiere a la transferencia del control de programa de un programa que corre en cierto momento a otro programa de servicio, como resultado de una solicitud generada en forma externa o interna. El control retorna al programa original después de que se ejecuta el programa de servicio.

El procedimiento de interrupción es, en principio, muy similar a una solicitud de subrutina, excepto por tres diferencias: 1) por lo general la interrupción se inicia mediante una señal externa o interna más que por la ejecución de una instrucción (excepto para una interrupción de programa, como se explica más adelante); 2) la dirección del programa de servicio de interrupción la determina la circuitería y no el campo de dirección de una instrucción; y 3) un procedimiento de interrupción por lo general almacena toda la información necesaria para definir el estado de la CPU en lugar de sólo almacenar el contador de programa. Estos tres conceptos de procedimiento se detallan en seguida.

Después de que se ha interrumpido un programa y se ha ejecutado la rutina de servicio, la CPU retorna al mismo estado exacto que tenía cuando ocurrió la interrupción. Sólo de esta manera el programa interrumpido podrá reanudar exactamente como si nada hubiera ocurrido. El estado de la CPU al final del ciclo de ejecución (cuando se reconoce la interrupción) está determinado por:

1. El contenido del contador del programa.
2. El contenido de todos los registros del procesador.
3. El contenido de ciertas condiciones de estado.

El conjunto de todas las condiciones de bit de estado en la CPU se denomina en ocasiones una *palabra de estado de programa* o PSW (*program status word*). La PSW se almacena en un registro de hardware separado y contiene la información de estado que caracteriza a la CPU. Normalmente, incluye los bits de estado de la última operación de la ALU y especifica las interrupciones que se han permitido ocurrir y si la CPU está operando en

pila para recuperar la PSW anterior y la dirección de retorno. La PSW se transfiere al registro de estado y la dirección de retorno al contador de programa. Por lo tanto, se restablece el estado de la CPU y el programa original puede continuar su ejecución.

Tipos de interrupciones

Existen tres tipos principales de interrupciones que producen una detención en la ejecución normal de un programa. Se clasifican de la manera siguiente:

1. Interrupciones externas
2. Interrupciones internas
3. Interrupciones de programa

Las interrupciones externas provienen de dispositivos de entrada y salida (E/S), de un dispositivo de temporización, de un circuito que monitorea la fuente de alimentación o de cualquier otra fuente externa. Algunos ejemplos de lo que produce interrupciones externas son dispositivos de E/S que solicitan transferencia de datos, dispositivos de E/S que terminan transferencia de datos, tiempo transcurrido de un evento o una falla de energía. Puede ocurrir una interrupción por tiempo transcurrido de un programa que esté en un ciclo que no termina y que, por lo tanto, excede su tiempo asignado. Una interrupción por falla de energía puede tener como su rutina de servicio a un programa que transfiere el estado completo de la CPU a una memoria no volátil en los pocos segundos anteriores a una falla de energía.

Las interrupciones internas surgen debido a la utilización ilegal o errónea de una instrucción o datos. Las interrupciones internas también se llaman *trampas*. Algunos ejemplos de las interrupciones provocadas por condiciones de error internas son los sobreflujos de registro, intentar dividir entre cero, un código de operación no válido, desbordamiento de pila, y violación de la protección. Por lo general, estas condiciones de error ocurren como resultado de una terminación prematura de la ejecución de una instrucción. El programa de servicio que procesa la interrupción interna determina la medida correctiva que se debe tomar.

La diferencia entre las interrupciones interna y externa es que la interna se inicia por alguna condición excepcional causada por el programa mismo, más bien que por un evento externo. Las interrupciones internas son sincrónicas con el programa en tanto que las externas no lo son. Si el programa se vuelve a ejecutar, las interrupciones internas ocurrirán en el mismo lugar cada vez. Las interrupciones externas dependen de condiciones independientes al programa que se ejecuta en ese momento.

Las interrupciones internas y externas se inician a partir de señales que ocurren en la circuitería de la CPU. Una interrupción de programa se inicia al ejecutar una instrucción. La interrupción de programa es una instrucción de solicitud especial que se comporta como una interrupción más que como

Una característica de los procesadores RISC es su capacidad para ejecutar una instrucción por ciclo de reloj. Esto se logra al hacer simultáneamente las fases de recuperación, decodificación y ejecución de dos o tres instrucciones, usando un procedimiento denominado paralelismo. Una instrucción cargar o almacenar puede requerir dos ciclos de reloj porque el acceso a memoria toma más tiempo que las operaciones de registro. En ocasiones se atribuyen a RISC un paralelismo eficiente y otras cuantas características, aunque también pueden existir en arquitecturas no RISC. Otras características que se le atribuyen a la arquitectura RISC son:

1. Una cantidad de registros en el procesador relativamente grande.
2. Uso de ventanas de registros traslapados para acelerar la llamada y retorno de procedimientos.
3. Paralelismo de instrucciones eficiente.
4. Soporte de compilador para traducción eficiente de programas de lenguaje de alto nivel a programas de lenguaje de máquina.

Es conveniente tener una gran cantidad de registros para almacenar los resultados intermedios y para optimizar las referencias de operandos. La ventaja del almacenamiento en registros, a diferencia del almacenamiento en memoria, es que los registros pueden transferir información a otros registros mucho más rápido que la transferencia de información hacia y desde la memoria. Por esa razón, se pueden minimizar las operaciones al conservar en registros los operandos a los que se accede con mayor frecuencia. Los estudios que muestran un rendimiento mejorado para la arquitectura RISC no establecen la diferencia entre los efectos del conjunto reducido de instrucciones y los efectos de un archivo de registros grande. Debido a eso, a veces se asocian con procesadores RISC una gran cantidad de registros en la unidad de procesamiento. En la siguiente sección se explica el uso de ventanas de registros traslapados cuando se transfiere el control de programas después de una solicitud de procedimiento. El conducto de instrucciones en RISC se presenta en la sección 9-5, después de que se explica el concepto de conducción y saturación de línea.

Ventanas de registros traslapados

En los lenguajes de programación de alto nivel ocurren con frecuencia llamadas y retornos de procedimientos. Cuando se traduce a lenguaje de máquina, una llamada de procedimiento produce una secuencia de instrucciones que guardan valores de registros, pasan parámetros que necesita el procedimiento y, después, solicitan una subrutina que ejecute el cuerpo del procedimiento. Después de un retorno de procedimiento, el programa restablece los valores de registro anteriores, trasmite los resultados al programa solicitante y retorna de la subrutina. Guardar y restablecer registros, y pasar parámetros y resultados son operaciones que consumen tiempo. Algunas

computadoras proporcionan bancos de registros múltiples y a cada procedimiento se le asigna su propio banco de registros. Esto elimina la necesidad de guardar y restablecer los valores de los registros. Algunas computadoras utilizan la pila de memoria para almacenar los parámetros que necesita el procedimiento, pero esto requiere un acceso a memoria cada vez que se accesa la pila.

Una característica de algunos procesadores RISC es que utilizan *ventanas de registros traslapados* para ofrecer el paso de parámetros y evitar la necesidad de guardar y restablecer valores de registros. Cada solicitud de procedimiento produce la asignación de una nueva ventana, que consiste en un conjunto de registros del archivo de registros, para que la use el nuevo procedimiento. Cada solicitud de procedimiento activa una nueva ventana de registros al incrementar un apuntador, mientras que el enunciado de retorno decrementa el apuntador y produce la activación de la ventana anterior. Las ventanas para procedimientos adyacentes tienen registros traslapados que comparten para proporcionar el paso de parámetros y resultados.

El concepto de ventanas con registros traslapados se ilustra en la figura 8-9. El sistema tiene un total de 64 registros. Los registros del R0 al R9 son registros globales que contienen parámetros que comparten todos los procedimientos. Los otros 64 registros se dividen en cuatro ventanas para alojar los procedimientos A, B, C y D. Cada ventana de registros consta de 10 registros locales y dos conjuntos de seis registros comunes a las ventanas adyacentes. Los registros locales se utilizan para las variables locales. Los registros comunes se utilizan para el intercambio de parámetros y resultados entre procedimientos adyacentes. Los registros traslapados comunes permiten que se pasen parámetros y el movimiento real de datos. En cualquier momento dado, sólo está activa una ventana de registro con un apuntador que la señala como activa. Cada solicitud de procedimiento activa una nueva ventana de registros al incrementar el apuntador. Los registros altos del procedimiento llamado se sobreponen a los registros bajos del procedimiento llamado, y por lo tanto, los parámetros se transfieren en forma automática del procedimiento que llama al llamado.

Como ejemplo, supongamos que el procedimiento A solicita el procedimiento B. Los registros del R26 al R31 son comunes a los dos procedimientos y, por lo tanto, el procedimiento A almacena los parámetros para el procedimiento B en estos registros. El procedimiento B utiliza los registros locales del R32 al R41 para el almacenamiento de variables locales. Si el procedimiento B solicita el procedimiento C, pasará los parámetros por medio de los registros del R42 al R47. Cuando el procedimiento B está listo para retornar al final su computación, el programa almacena los resultados de estos cálculos en los registros del R26 al R31 y transfiere de vuelta la ventana de registros al procedimiento A. Notese que los registros R10 al R15 son comunes a los procedimientos A y D, por lo que las cuatro ventanas tienen una organización circular en donde A es adyacente a D.

31	24 23	19 18	14 13 12	5 4	0
Código de operación	Rd	Rs	0	Sin usar	S2
8	5	5	1	8	5

a) Modo de registro: (S2 especifica un registro)

31	24 23	19 18	14 13 12	0
Código de operación	Rd	Rs	1	S2
8	5	5	1	13

b) Modo de registro inmediato: (S2 especifica un operando)

31	24 23	19 18	14 13 12	0
Código de operación	COND		Y	
8	5	5	19	

c) Modo relativo a PC:

Figura 8-10 Formatos de instrucciones de la RISC I de Berkeley.

los bits en el código de operación especifican una operación y el octavo indica si se actualizan los bits de estado después de una operación ALU. Para instrucciones de registro a registro, el campo Rd de 5 bits selecciona uno de los 32 registros como destino para el resultado de la operación. La operación se ejecuta con los datos especificados en los campos Rs y S2. Rs es uno de los registros fuente. Si el bit 13 de la instrucción es 0, los 5 bits de orden menor de S2 especifican otro registro fuente. Si el bit 13 de la instrucción es 1, S2 especifica una constante de 13 bits de signo extendido. Por lo tanto, la instrucción tiene un formato de tres direcciones, pero la segunda fuente puede ser un registro o un operando inmediato. Las instrucciones de acceso a memoria utilizan Rs para especificar una dirección de 32 bits en un registro y S2 para especificar un desplazamiento. El registro R0 contiene sólo números 0, por lo que puede utilizarse en cualquier campo para especificar una cantidad de cero. El tercer formato de instrucción combina los tres últimos campos para formar una dirección relativa de diecinueve bits Y y se usa sobre todo con las instrucciones brincar y llamar. El campo COND sustituye el campo Rd para instrucciones de brinco y se utiliza para especificar una de 16 condiciones de brinco posibles.

Las 31 instrucciones de la RISC I se listan en la tabla 8-12. Se han agrupado en tres categorías. Las instrucciones de manipulación de datos ejecutan operaciones aritméticas, lógicas y de corrimiento. Los símbolos bajo las columnas del código de operación y operando se utilizan cuando se escri-

- 8-5.** Sea $SP = 000000$ en la pila de la figura 8-3. ¿Cuántos datos hay en la pila si:
- ¿LLENO = 1 y VACIO = 0?
 - ¿LLENO = 0 y VACIO = 1?
- 8-6.** Una pila está organizada de manera que SP siempre apunte a la siguiente posición vacía en la pila. Esto significa que SP puede inicializarse a 4000 en la figura 8-4 y el primer dato en la pila se almacena en la posición 4000. Liste las microoperaciones para las operaciones push y pop.
- 8-7.** Convierta las siguientes expresiones aritméticas de notación interna fija a polaca inversa.
- $A * B + C * D + E * F$
 - $A * B + A * (B * D + C * E)$
 - $A + B * [C * D + E * (F + G)]$
 - $$\frac{A * [B + C * (D + E)]}{F * (G + H)}$$
- 8-8.** Convierta las siguientes expresiones aritméticas de notación polaca inversa a interna fija.
- $A \ B \ C \ | \ D \ E \ + \ * \ - \ /$
 - $A \ B \ C \ D \ E \ * \ / \ - \ +$
 - $A \ B \ C \ * \ / \ D \ - \ E \ F \ / \ +$
 - $A \ B \ C \ D \ E \ F \ G \ + \ * \ + \ * \ + \ *$
- 8-9.** Convierta la siguiente expresión numérica aritmética a notación polaca inversa y muestre las operaciones de pila para evaluar el resultado numérico.
- $$(3 + 4)[10(2 + 6) + 8]$$
- 8-10.** Un sistema primero en entrar, primero en salir (FIFO) tiene una organización de memoria que almacena información de manera que el dato que se almacena primero es el primero que se recupera. Muestre cómo opera una memoria FIFO con tres contadores. Un contador de escritura WC contiene la dirección para escribir en la memoria. Un contador de lectura RC contiene la dirección para leer de la memoria. Un contador de almacenamiento disponible ASC indica la cantidad de palabras almacenadas en FIFO. ASC se incrementa para cada palabra almacenada y se decrementa para cada dato recuperado.
- 8-11.** Una computadora tiene instrucciones de 32 bits y direcciones de 12 bits. Si hay 250 instrucciones de dos direcciones, ¿cuántas instrucciones de una dirección pueden formularse?
- 8-12.** Escriba un programa para evaluar el enunciado aritmético:

$$X = \frac{A - B + C * (D * E - F)}{G + H * K}$$

registro de procesador R1 contiene el número 200. Evalúe la dirección efectiva si el modo de direccionamiento de la instrucción es: a) directo; b) inmediato c) relativo; d) indirecto por registro; e) indexado con R1 como el registro índice.

- 8-19. Considerando una computadora de 8 bits, muestre la suma de precisión múltiple de los dos números sin signo de 32 bits que se listan en seguida, utilizando la suma con instrucciones de acarreo. Cada byte se expresa como un número hexadecimal de dos dígitos.

$$(6E\ C3\ 56\ 7A) + (13\ 55\ 6B\ 8F)$$

- 8-20. Realice la lógica AND, OR, y XOR con las dos series binarias 10011100 y 10101010.
- 8-21. Dado el valor de 16 bits 1001101011001101. ¿Qué operación debe ejecutarse con el fin de:
- Borrar a 0 los primeros ocho bits?
 - Activar a 1 los últimos ocho bits?
 - Complementar los ocho bits de en medio?
- 8-22. Un registro de 8 bits contiene el valor 01111011 y el bit de acarreo es igual a 1. Ejecute las ocho operaciones de corrimiento proporcionadas por las instrucciones que se listan en la tabla 8-9. Cada vez, inicie del valor inicial que se dio en esta pregunta.
- 8-23. Represente los siguientes números con signo en binario utilizando ocho bits. +83; -83; +68; -68.
- Realice la suma $(-83) + (+68)$ en binario e interprete el resultado obtenido.
 - Realice la resta $(-68) - (+83)$ en binario e indique si hay sobreflujo.
 - Recorra a la derecha el binario -68 una vez y proporcione el valor del número recorrido en decimal.
 - Recorra a la izquierda el binario -83 e indique si hay un sobreflujo.
- 8-24. Demuestre que el circuito etiquetado "verificar si hay una salida cero" en la figura 8-8 es una compuerta NOR de 8 bits.
- 8-25. Una computadora de 8 bits tiene un registro R. Determine los valores de los bits de estado C, S, Z y V (figura 8-8) después de cada una de las siguientes instrucciones. El valor inicial del registro R, en cada caso es el hexadecimal 72. Los números siguientes también están en hexadecimal.
- Sume el operando inmediato C6 a R.
 - Sume el operando inmediato 1E a R.
 - Reste el operando inmediato 9A de R.
 - Aplique la función AND del operando inmediato 8D a R.
 - Aplique la función OR exclusiva R con R.
- 8-26. Dos números sin signo A y B se comparan al restar $A - B$. El bit-de estado de acarreo se considera como un bit de préstamo después de una instrucción comparar en la mayoría de las computadoras comerciales, por lo tanto C =

9. Mano, M. M., *Computer Engineering: Hardware Design*. Englewood Cliffs, NJ: Prentice Hall, 1988.
10. Murray, W. D., *Computer and Digital System Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1990.
11. Patterson, D. A., y J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann Publishers, 1990.
12. Patterson, D. A., y C. H. Sequin, "A VLSI RISC." *IEEE Computer*, Septiembre de 1982, pp. 8-22.
13. Pollard, L. H., *Computer Design and Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1990.
14. Rafiquzzaman, M., y R. Chandra, *Modern Computer Architecture*. St. Paul, MN: West Publishers, 1988.
15. Siewiorek, D., C. G. Bell y A. Newell, *Computer Structures: Principles and Examples*. Nueva York: McGraw-Hill, 1982.
16. Stallings, W., *Computer Organization and Architecture*, 2a. Ed. Nueva York, Macmillan, 1989.
17. Tanenbaum, A. S., *Structured Computer Organization*, 3a. Ed. Englewood Cliffs, NJ: Prentice Hall, 1990.
18. Tomek, I., *Introduction to Computer Organization*. Rockville, MD: Computer Science Press, 1981.
19. Toy, W., y B. Zee, *Computer Hardware/Software Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1986.
20. Ward, S. A. y R. H. Halstead, Jr., *Computation Structures*. Cambridge, MA: MIT Press, 1990.

EN ESTE CAPÍTULO

- 9-1 Procesamiento paralelo
- 9-2 Arquitectura paralela
- 9-3 Línea paralela aritmética
- 9-4 Línea paralela de instrucciones
- 9-5 Arquitectura paralela RISC
- 9-6 Procesamiento vectorial
- 9-7 Arreglo de procesador SIMD

9-1 Procesamiento paralelo

El procesamiento paralelo es un término que se usa para denotar un grupo de técnicas significativas que se usan para proporcionar tareas simultáneas de procesamiento de datos con el fin de aumentar la velocidad computacional de un sistema de computadora. En lugar de procesar cada instrucción en forma secuencial como en una computadora convencional, un sistema de procesamiento paralelo puede ejecutar procesamiento concurrente de datos para conseguir un menor tiempo de ejecución. Por ejemplo, cuando se ejecuta una instrucción en la ALU, puede leerse la siguiente instrucción de la memoria. El sistema puede tener dos o más ALUS y ser capaz de ejecutar dos o más instrucciones al mismo tiempo. Además, el sistema puede tener dos o más procesadores operando en forma concurrente. El propósito del procesamiento paralelo es acelerar las posibilidades de procesamiento de la computadora y aumentar su eficiencia, esto es, la cantidad de procesamiento que puede lograrse durante un cierto intervalo de tiempo. La cantidad de circuitería aumenta con el procesamiento paralelo y, con él, también el costo del sistema. Sin embargo, los descubrimientos tecnológicos han reducido el costo de la circuitería a un punto en donde las técnicas de procesamiento paralelo son económicamente factibles.

eficiencia

Conforme aumenta el número de tareas, la aceleración se acercará a 4, lo cual es igual al número de segmentos en el conducto. Si consideramos que $t_n = 60$ ns, la aceleración se torna $60/20 = 3$.

Para duplicar la ventaja de velocidad teórica de un proceso paralelo mediante unidades funcionales múltiples, es necesario construir k unidades idénticas que operarán en paralelo. Esto implica que puede esperarse que un procesador de arquitectura paralela de k segmentos iguale el desempeño de k copias de un circuito de arquitectura no paralela equivalente bajo condiciones de operación iguales. Esto se ilustra en la figura 9-5, en donde están conectados en paralelo cuatro circuitos idénticos. Cada circuito P ejecuta la misma tarea de un circuito con arquitectura paralela equivalente. En lugar de operar con los datos de entrada en secuencia, como en una línea, los circuitos paralelos aceptan cuatro conjuntos de datos de entrada en forma simultánea y ejecutan cuatro tareas al mismo tiempo. En lo que se refiere a la velocidad de operación esta es equivalente a una línea con cuatro segmentos. Nótese que el circuito de cuatro unidades de la figura 9-5, constituye una organización de instrucción única, datos múltiples (SIMD), porque se utiliza la misma instrucción para operar sobre datos múltiples en paralelo.

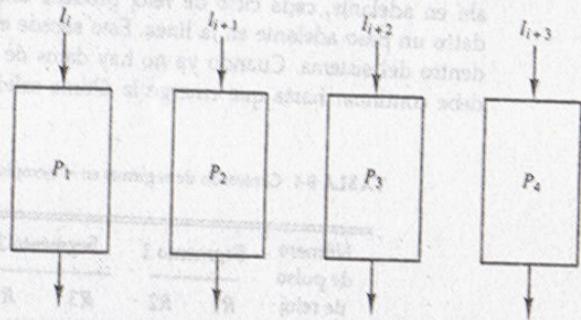


Figura 9-5 Unidades funcionales múltiples en paralelo.

Existen varias razones por las que una arquitectura paralela no puede operar a su máxima velocidad teórica. Los diferentes segmentos pueden requerir tiempos diferentes para completar su suboperación. Debe elegirse el ciclo de reloj para que iguale el tiempo de retraso del segmento con el máximo tiempo de propagación. Esto provoca que los otros segmentos desperdicien tiempo mientras esperan el siguiente ciclo de reloj. Además no siempre es correcto considerar que un circuito de arquitectura tiene el mismo retraso de tiempo que un circuito con arquitectura paralela equivalente. Muchos de los registros intermedios no se necesitarán en un circuito de una sola unidad el cual, por lo general, puede construirse por completo como un circuito combinatorio. No obstante, la técnica de arquitectura paralela proporciona una operación más rápida que una secuencia puramente serial, aunque nunca se logra por completo la máxima velocidad teórica.

Existen dos áreas de diseño de computadoras en las que es aplicable la organización paralela. Una *línea aritmética* divide una operación aritmética en suboperaciones que se ejecutan en los segmentos de la línea. Una *línea de instrucciones* opera sobre un flujo de instrucciones al sobreponer las fases de recuperación, decodificación y ejecución del ciclo de instrucción. En las siguientes secciones se explican los dos tipos de líneas.

9-3 Línea paralela aritmética

Por lo general, se encuentran unidades aritméticas de arquitectura paralela en computadoras de gran velocidad. Se usan para implantar operaciones de punto flotante, multiplicación de números de punto fijo y cálculos similares encontrados en problemas científicos. En esencia, un multiplicador de arquitectura paralela es un arreglo multiplicador como el que se describe en la figura 10-10, con sumadores especiales diseñados para minimizar el tiempo de propagación de acarreo mediante los productos parciales. Las operaciones de punto flotante se descomponen con facilidad en suboperaciones, como se muestra en la sección 10-5. Ahora mostraremos un ejemplo de una unidad paralela para suma y resta de punto flotante.

Las entradas para la línea sumadora de punto flotante son dos números binarios normalizados de punto flotante.

$$X = A \times 2^a$$

$$Y = B \times 2^b$$

A y B son dos fracciones que representan las mantisas, en tanto que a y b son los exponentes. Puede ejecutarse la suma y resta de punto flotante en cuatro segmentos, como se muestra en la figura 9-6. Los registros etiquetados R se colocan entre los segmentos para almacenar los resultados intermedios. Las suboperaciones que se ejecutan en los cuatro segmentos son:

1. Comparar los exponentes.
2. Alinear las mantisas.
3. Sumar o restar las mantisas.
4. Normalizar el resultado.

Esto se apega al procedimiento delineado en el diagrama de flujo de la figura 10-15, pero se usan algunas variaciones para reducir el tiempo de ejecución de las suboperaciones. Se comparan los exponentes al restarlos para determinar su diferencia. Se escoge el exponente mayor como el exponente del resultado. La diferencia entre exponentes determina cuántas veces debe ejecutarse un corrimiento a la derecha sobre la mantisa asociada con el exponente menor. Esto provoca un alineamiento de las dos mantisas. Debe notarse que el corrimiento debe estar diseñado como un circuito combina-

o punto se realizan con los exponentes de ambos; si ambos están iguales se obtiene la resta mediante la operación $b - a$; si no, se ajusta el exponente del menor para que ambos tengan el mismo exponente, se multiplican los resultados y se suman los exponentes.

En la figura 9-6 se muestra la arquitectura paralela para la suma y resta de punto flotante. Se observa que la diferencia entre los exponentes es la cantidad que se resta en el primer segmento. La diferencia se usa para ajustar el exponente del menor número y para desplazar la parte fraccionaria del resultado. La parte fraccionaria se alinea en el segundo segmento y se suman o restan las partes fraccionarias. El resultado se normaliza en el cuarto segmento y se ajusta el exponente en el tercero.

La arquitectura paralela para la multiplicación y división de punto flotante es similar, pero se requiere un solo segmento para multiplicar o dividir los exponentes y otro para multiplicar o dividir las mantis.

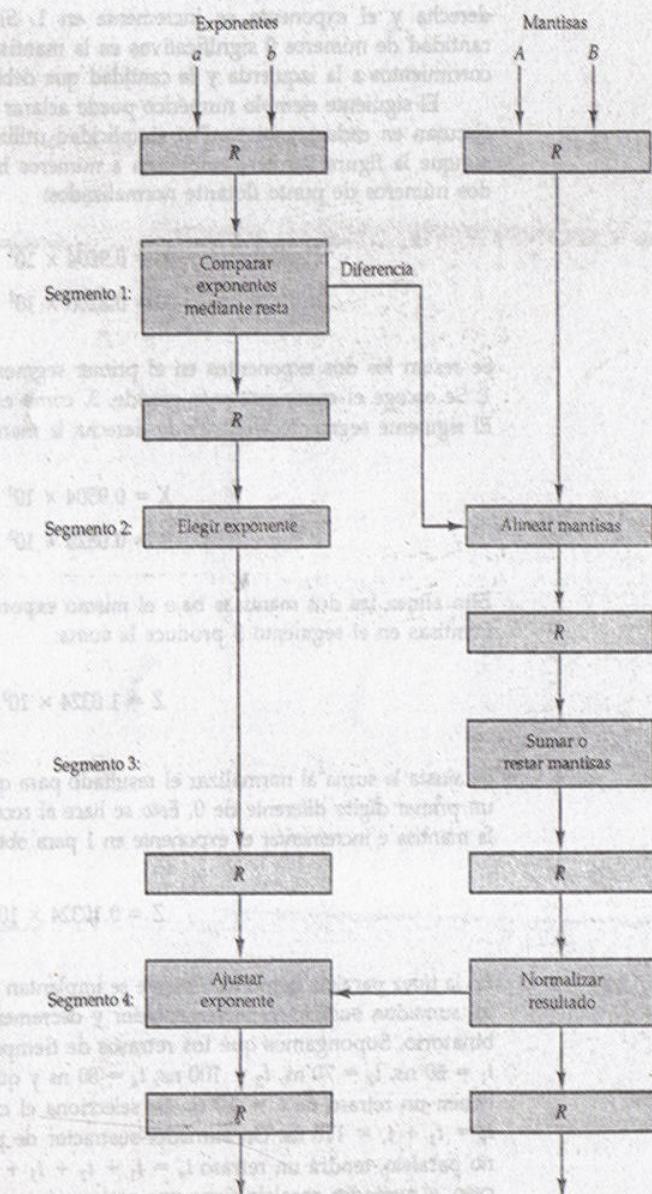


Figura 9-6 Arquitectura paralela para suma y resta de punto flotante.

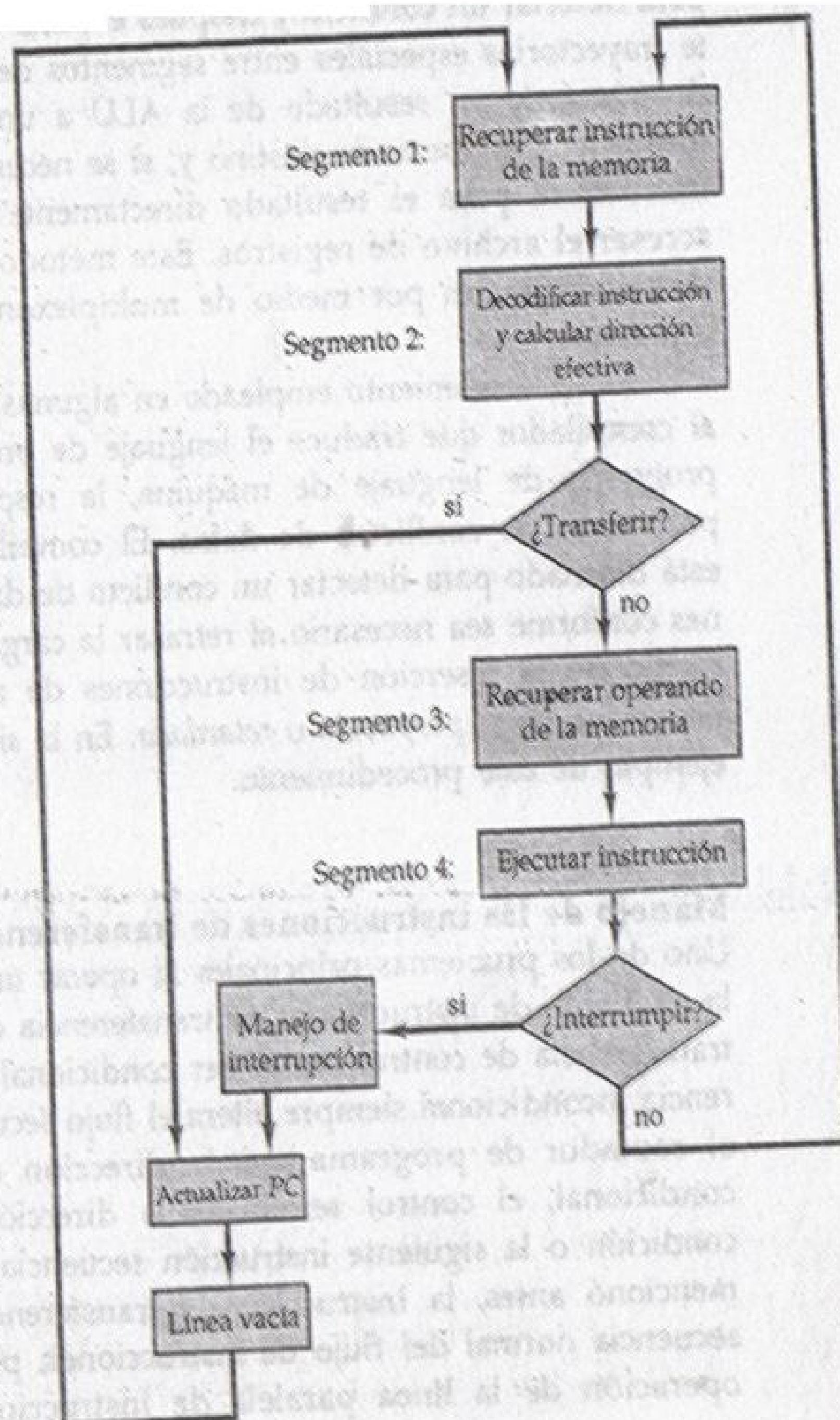
dos canales de memoria para accesar las instrucciones y los datos en módulos separados. De esta manera, pueden leerse en forma simultánea una palabra de instrucción y una palabra de datos de dos módulos diferentes.

El diseño de una línea paralela de instrucciones será más eficiente si se divide el ciclo de instrucciones en segmentos de igual duración. El tiempo que necesita cada paso para terminar su función depende de la instrucción y de la manera como se ejecuta.

Ejemplo: Línea paralela de instrucciones de cuatro segmentos

Consideremos que pueden combinarse en un segmento la decodificación de la instrucción y el cálculo de la dirección efectiva. Además, consideremos que la mayoría de las instrucciones colocan el resultado en un registro de

Figura 9-7 Arquitectura paralela de CPU de cuatro segmentos.



arquitectura paralela emplean diferentes técnicas de hardware para minimizar la degradación del desempeño provocada por las instrucciones de transferencia de control.

Una manera de manejar una transferencia condicional es recuperar con anticipación la dirección objetivo, además de la instrucción que sigue a la transferencia. Ambas se guardan hasta que se ejecuta la transferencia. Si la condición de la transferencia se cumple, la línea paralela continúa a partir de la instrucción objetivo de la transferencia. Una extensión de este procedimiento es continuar la recuperación de instrucciones de ambos lugares hasta que se tome la decisión de la transferencia. En ese momento, el control selecciona el flujo de instrucciones de la secuencia de programa correcta.

Otra posibilidad es usar un *registro de ramificación objetivo* (branch target buffer, BTB). El BTB es una memoria asociativa (véase la sección 12-4) incluida en el segmento de recuperación de la línea. Cada entrada en el BTB consiste en la instrucción de transferencia recuperada con anticipación y la instrucción objetivo para esa transferencia. También almacena unas cuantas instrucciones que se encuentran después de la instrucción objetivo de la transferencia. Cuando la línea decodifica una instrucción de transferencia, busca la memoria asociativa BTB para la dirección de la instrucción. Si está en el BTB, se puede disponer de la instrucción en forma directa y la recuperación con anticipación continúa a partir de la nueva trayectoria. Si la instrucción no está en el BTB, la línea paralela cambia a un nuevo ciclo de instrucciones y almacena la instrucción objetivo en el BTB. La ventaja de este esquema es que las instrucciones de transferencia de control que han ocurrido con anticipación están disponibles con facilidad en la línea, sin interrupciones.

Una variante del BTB es el *registro de ciclo*. Este es un pequeño archivo de registros, de muy alta velocidad, que se mantiene en el segmento de recuperación de instrucciones de la línea paralela. Cuando se detecta un ciclo de programa, se almacena por completo en el registro de ciclo, incluyendo todas las transferencias. El ciclo de programa puede ejecutarse en forma directa, sin tener que accesar la memoria hasta que se retira el modo de ciclo con la transferencia de control final.

Otro procedimiento que utilizan algunas computadoras es la *predicción de transferencia de control y línea paralela*. Una arquitectura paralela con predicción de transferencia de control utiliza cierta lógica adicional para predecir el resultado de una instrucción de transferencia condicional antes de que se execute. Después, la línea paralela empieza a recuperar con anticipación el flujo de instrucciones de la trayectoria predicha. Una predicción correcta elimina el tiempo perdido causado por el proceso de transferencia de control.

Un procedimiento usado en la mayoría de los procesadores RISC es la *transferencia pospuesta o retardada*. En este procedimiento, el procesador detecta las instrucciones de transferencia de control y reordena la secuencia de código de lenguaje de máquina al insertar instrucciones útiles que conservan sin interrupciones la operación de la línea paralela. Un ejemplo de transfe-

rencia retardada es la inserción de una instrucción de no operación después de una instrucción de transferencia de control. Esto hace que la computadora recupere la dirección objetivo, durante la ejecución de la instrucción de no operación, lo que permite tener un flujo continuo del conducto. En la siguiente sección se presenta un ejemplo de transferencia de control pospuesta.

9-5 Arquitectura paralela RISC

En la sección 8-8 se presentó la computadora con conjunto reducido de instrucciones (RISC). Entre las características que se le atribuyen a RISC está la capacidad de usar un conjunto de instrucciones eficiente. Puede usarse la sencillez del conjunto de instrucciones para implantar una línea paralela de instrucciones con un número pequeño de operaciones y que cada una se ejecute en un ciclo de reloj. Debido al formato de instrucciones de longitud fija, la decodificación de la operación puede ocurrir al mismo tiempo que la selección de registro. Todas las instrucciones de manipulación de datos tienen operaciones registro a registro. Como todos los operandos están en registros, no es necesario calcular una dirección efectiva o recuperar operandos de la memoria. Por lo tanto, la línea paralela de instrucciones puede implantarse con dos o tres segmentos. Un segmento recupera la instrucción de la memoria del programa y el otro segmento ejecuta la instrucción en la ALU. Puede usarse un tercer segmento para almacenar el resultado de la operación de la ALU en un registro destino.

En la arquitectura RISC, las instrucciones de transferencia de datos están limitadas a las instrucciones cargar y almacenar. Estas instrucciones utilizan un direccionamiento indirecto por registros. Por lo general, necesitan tres o cuatro etapas en el segmento. Para evitar conflictos entre un acceso a memoria con el fin de recuperar una instrucción y cargar o almacenar un operando, gran parte de las máquinas RISC utilizan dos canales separados con dos memorias: una para almacenar las instrucciones y otra para almacenar los datos. En algún momento, las dos memorias pueden operar a la misma velocidad que el reloj de la CPU y se denominan memorias caché (véase la sección 12-6).

Como se mencionó en la sección 8-8, una de las principales ventajas de la RISC es su capacidad para ejecutar instrucciones a una velocidad de una por ciclo de reloj. No es posible esperar que cada instrucción se recupere de la memoria y se ejecute en un ciclo de reloj. En realidad, lo que se hace es empezar cada instrucción con cada ciclo de reloj y mediante su arquitectura paralela el procesador logra el propósito de una ejecución de instrucción en un solo ciclo. La ventaja de la RISC sobre la CISC (computadora con conjunto de instrucciones complejo) es que la RISC puede tener segmentos paralelos, al requerir sólo un ciclo de reloj, en tanto la SISC utiliza muchos segmentos en su arquitectura paralela y el segmento más grande requiere dos o más ciclos de reloj.

soporte de
compilador

Otra característica de la RISC es el soporte que proporciona el compilador que traduce el programa de lenguaje de alto nivel en un programa de lenguaje de máquina. En lugar de diseñar hardware para manejar las dificultades asociadas con conflictos de datos y pérdidas por transferencias de control, los procesadores RISC se basan en la eficiencia del compilador para detectar y minimizar los retrasos que se encuentran con estos problemas. Los siguientes ejemplos muestran cómo un compilador puede optimizar el programa del lenguaje de máquina para compensar los conflictos de la arquitectura paralela.

Ejemplo: Línea paralela de instrucciones de tres segmentos

Un conjunto típico de instrucciones para un procesador RISC se enlista en la tabla 8-12. En esta tabla apreciamos que existen tres tipos de instrucciones. Las instrucciones de manipulación de datos operan sobre los datos en los registros del procesador. Las instrucciones de transferencia de datos son instrucciones de carga y almacenamiento que utilizan una dirección efectiva obtenida de la suma del contenido de dos registros o de un registro y un desplazamiento constante proporcionado a la instrucción. Las instrucciones de control de transferencia del programa utilizan valores de registro y una constante para evaluar la dirección de transferencia, la cual se transfiere a un registro o al contador de programa PC.

Consideremos ahora la operación de hardware para dicha computadora. La sección de control recupera la instrucción de la memoria del programa dentro de un registro de instrucción. La instrucción se decodifica al mismo tiempo que se seleccionan los registros necesarios para la ejecución de la instrucción. La unidad del procesador consta de un número de registros y una unidad aritmética lógica (ALU) que ejecuta las operaciones aritméticas, lógicas y de corrimiento. Se usa una memoria de datos para cargar o almacenar los datos de un registro seleccionado en el archivo de registros. El ciclo de instrucción puede dividirse en tres suboperaciones e implantarse en tres segmentos:

I: Recuperación de instrucción

A: Operación de la ALU

E: Ejecutar instrucción

El segmento I recupera la instrucción de la memoria del programa. La instrucción se decodifica y se ejecuta una operación ALU en el segmento A. La ALU se utiliza para tres funciones diferentes, dependiendo de la instrucción codificada. Ejecuta una operación para una instrucción de manipulación de datos, evalúa la dirección efectiva para una instrucción de carga o almacenamiento o calcula la dirección de transferencia para una instrucción de transferencia de control del programa. El segmento E dirige la salida de la ALU a uno de tres destinos, dependiendo de la instrucción decodificada.

Un ejemplo de transferencia retardada se muestra en la figura 9-10. El programa para este ejemplo consta de cinco instrucciones:

- Cargar de la memoria a $R1$.
- Incrementar $R2$.
- Sumar $R3$ a $R4$.
- Restar $R5$ de $R6$.
- Transferir el control del programa a la dirección X .

En la figura 9-10(a) el compilador inserta dos instrucciones no-op después de la transferencia de control del programa. La dirección de transferencia X se carga en PC en el ciclo de reloj 7. Se pospone la recuperación de la instrucción en X en dos ciclos de reloj mediante las instrucciones no-op. La instrucción en X empieza la fase de recuperación en el ciclo de reloj 8, después de que se ha actualizado el contador de programa PC .

Figura 9-10 Ejemplo de transferencia retardada.

Ciclos de reloj:	1	2	3	4	5	6	7	8	9	10
1. Cargar	I	A	E							
2. Incrementar	I	A	E							
3. Sumar			I	A	E					
4. Restar				I	A	E				
5. Transferir a X					I	A	E			
6. No operación						I	A	E		
7. No operación							I	A	E	
8. Instrucción en X							I	A	E	

a) Usando instrucciones de no operación

Ciclos de reloj:	1	2	3	4	5	6	7	8
1. Cargar	I	A	E					
2. Incrementar		I	A	E				
3. Transferir a R			I	A	E			
4. Sumar				I	A	E		
5. Restar					I	A	E	
6. Instrucción en X						I	A	E

b) Reordenando las instrucciones

El programa en la figura 9-10(b) se vuelve a arreglar al colocar las instrucciones de sumar o restar después de la instrucción de transferencia de control y no antes, como en el programa original. Una revisión de la temporización de la línea paralela muestra que PC está actualizado al valor de X en el ciclo de reloj 5, pero las instrucciones de sumar y restar se recuperan de la memoria y se ejecutan en la secuencia deseada. En otras palabras, si la instrucción cargar está en la dirección 101 y X es igual a 350, la instrucción de transferencia de control se recupera de la dirección 103. La instrucción de sumar se recupera de la dirección 104 y se ejecuta en el ciclo de reloj 6. La instrucción de restar se recupera de la dirección 105 y se ejecuta en el ciclo de reloj 7. Como el valor de X se transfiere a PC con el ciclo de reloj 5 en el segmento E, la instrucción recuperada de la memoria en el ciclo de reloj 6 proviene de la dirección 350, la cual es la instrucción de la dirección de transferencia.

9-6 Procesamiento vectorial

Existe un tipo de problema computacional que está más allá de la capacidad de una computadora convencional. Este problema se caracteriza por el hecho de que requiere una gran cantidad de cálculos que le tomará a una computadora convencional días o incluso semanas para terminar. En muchas aplicaciones científicas y de ingeniería el problema puede formularse en términos de vectores y matrices que se prestan a procesamiento de vectores.

Las computadoras con posibilidades de procesamiento de vectores tienen demanda en aplicaciones especializadas. Las siguientes son áreas de aplicación representativas, en las cuales el procesamiento de vectores es de la mayor importancia.

- Predicciones climatológicas a largo plazo
- Exploraciones petrolíferas
- Análisis de datos sísmicos
- Diagnóstico médico
- Simulaciones de aerodinámica y vuelo espacial
- Sistemas expertos y de inteligencia artificial
- Mapeo de los genes humanos
- Procesamiento de imágenes

Sin computadoras de avanzado diseño, muchos de los cálculos requeridos no pueden llevarse a cabo dentro de un tiempo razonable. Para conseguir el nivel requerido de desempeño de calidad es necesario utilizar la circuitería más rápida y confiable y aplicar procedimientos innovadores de técnicas de procesamiento paralelo y de vectores.

control de programa y escalares se ejecutan en forma directa dentro de la unidad de control maestro. Las instrucciones de vector se transmiten a todos los PE en forma simultánea. Cada PE utiliza operandos almacenados en su memoria local. Los operandos de vector se distribuyen a las memorias locales antes de la ejecución en paralelo de la instrucción.

Por ejemplo, consideremos la suma de vector $C = A + B$. Primero, la unidad de control maestro almacena los iésimos componentes a_i y b_i de A y B en la memoria local M_i para $i = 1, 2, 3, 4, \dots, n$. Despues transmite la instrucción sumar de punto flotante $c_i = a_i + b_i$ a todos los PE, causando que la suma se realice en forma simultánea. Los componentes de c_i están almacenados en localidades fijas en cada memoria local. Esto produce la suma del vector deseado en un ciclo de suma.

Se usan esquemas de enmascaramiento, con el fin de controlar el estado de cada PE durante la ejecución de instrucciones de vector. Cada PE tiene una bandera que se activa cuando el PE está activo y se desactiva cuando está inactivo. Esto asegura que sólo están activos durante la ejecución de la instrucción aquellos PE que necesitan participar. Por ejemplo, supongamos que un arreglo de procesador contiene un conjunto de 64 PE. Si se va a procesar una longitud de vector menor que 64 conjuntos de datos, la unidad de control selecciona la cantidad correcta de PE que van a estar activos. La unidad de control debe dividir los vectores con longitud mayor de 64, en partes de 64 palabras.

El procesador de arreglo SIMD más conocido es la computadora ILLIAC IV, desarrollada en la Universidad de Illinois y fabricada por Burroughs Corp. Esta computadora ya no está en operación. Los procesadores SIMD son computadoras altamente especializadas. Sobre todo son convenientes para problemas numéricos que pueden expresarse en forma de vector o de matriz. Sin embargo, no son muy eficientes en otros tipos de cálculos o para trabajar con programas de procesamiento de datos convencionales.

PROBLEMAS

- 9-1.** En ciertos cálculos científicos es necesario ejecutar la operación aritmética $(A_i + B_j)(C_i + D_j)$ con una serie de números. Especifique una configuración para realizar esta tarea. Liste el contenido de todos los registros en la arquitectura paralela para $i = 1$ a 6.
- 9-2.** Dibuje un diagrama espacio tiempo para una arquitectura paralela de seis segmentos que muestre el tiempo que se necesita para procesar ocho tareas.
- 9-3.** Determine la cantidad de ciclos de reloj que se necesitan para procesar 200 tareas en una arquitectura paralela de seis segmentos.
- 9-4.** Un sistema no paralelo necesita 50 ns para procesar una tarea. La misma tarea puede procesarse en una arquitectura paralela de seis segmentos con

EN ESTE CAPÍTULO

- 10-1 Introducción
- 10-2 Suma y resta
- 10-3 Algoritmos de multiplicación
- 10-4 Algoritmos de división
- 10-5 Operaciones aritméticas de punto flotante
- 10-6 Unidad aritmética decimal
- 10-7 Operaciones aritméticas decimales

10-1 Introducción

Las instrucciones aritméticas en las computadoras digitales manipulan los datos para producir los resultados necesarios para la solución de problemas computacionales. Estas instrucciones ejecutan cálculos aritméticos y son responsables de la mayor parte de la actividad para el procesamiento de datos en una computadora. Las cuatro operaciones aritméticas básicas son suma, resta, multiplicación y división. De estas cuatro operaciones básicas es posible formular otras operaciones aritméticas y resolver problemas científicos mediante métodos de análisis numérico.

Un procesador aritmético es la parte de una unidad de procesador que ejecuta operaciones aritméticas. El tipo de datos que se considera residen en los registros del procesador durante la ejecución de una instrucción aritmética se especifica en la definición de la instrucción. Una instrucción aritmética puede especificar datos binarios o decimales y en cada caso los datos pueden estar en forma de punto fijo o flotante. Los números de punto fijo pueden representar enteros o fracciones. Los números negativos pueden estar en representación de un complemento de signo o de magnitud de signo. El procesador aritmético es muy simple si sólo se incluye una instrucción de



Figura 10-1 Hardware para suma y resta de magnitud con signo.

se suman A y B . El registro A proporciona otras microoperaciones que pueden necesitarse cuando especificamos la secuencia de pasos en el algoritmo.

La suma de A y B se realiza mediante el sumador paralelo. La salida S (suma) del sumador se aplica a la entrada del registro A . El complementador proporciona una salida de B o el complemento de B , dependiendo del estado del control de modo M . El complementador consta de compuertas OR exclusivas y el sumador paralelo se forma de circuitos sumadores completos, como se muestra en la figura 4-7, en el capítulo 4. La señal M se aplica también al acarreo de entrada del sumador. Cuando $M = 0$, la salida de B se transfiere al sumador, el acarreo de entrada es 0 y la salida del sumador es igual a la suma $A + B$. Cuando $M = 1$, el complemento a 1 de B se aplica al sumador, el acarreo de entrada es 1 y la salida $S = 1 A + B + 1$. Esto es igual a A más el complemento a 2 de B , lo cual es equivalente a la resta $A - B$.

Algoritmo de hardware

El diagrama de flujo para el algoritmo de hardware se presenta en la figura 10-2. Los dos signos A_s y B_s se comparan mediante una compuerta OR exclusiva. Si la salida de la compuerta es 0, los signos son idénticos; si es 1, los signos son diferentes. Para una operación de *sumar*, signos idénticos dictan que se sumen las magnitudes. Para una operación de *restar*, signos diferentes determinan que se sumen las magnitudes. Las magnitudes se suman con una microoperación $EA \leftarrow A + B$, donde EA es un registro que combina E y A . El acarreo en E después de la suma, constituye un sobreflujo si es igual a 1. El valor de E se transfiere al flip-flop AVF de sobreflujo de suma.

Se restan las dos magnitudes si los signos son diferentes para una operación de *sumar* o idénticos para una operación de *restar*. Se restan las

número binario en el contador A 6-01

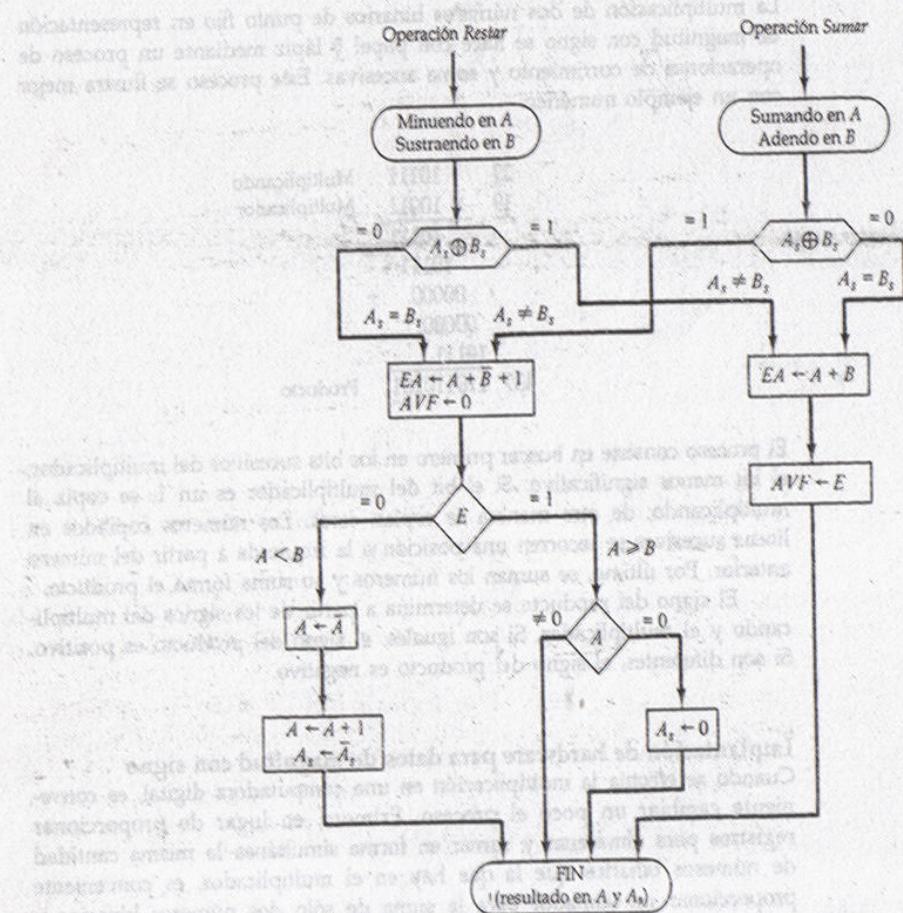


Figura 10-2 Diagrama de flujo para las operaciones sumar y restar.

magnitudes al sumar A al complemento a 2 de B. No puede ocurrir un sobrefluido, si se restan los números, de modo que AVF se borran a 0. Un 1 en E indica que $A \geq B$ y que el número en A es el resultado correcto. Si este número es 0, el signo A_1 debe hacerse positivo para evitar un cero negativo. Un cero en E indica que $A < B$. Para este caso es necesario tomar el complemento a 2 en A. Esta operación puede realizarse con una microoperación $A \leftarrow A + 1$. Sin embargo, consideraremos que el registro A tiene circuitos para las microoperaciones de *complementar* e *incrementar*, por lo que se obtiene el complemento a 2 a partir de estas dos microoperaciones. En

10-3 Algoritmos de multiplicación

La multiplicación de dos números binarios de punto fijo en representación de magnitud con signo se hace con papel y lápiz mediante un proceso de operaciones de corrimiento y suma sucesivas. Este proceso se ilustra mejor con un ejemplo numérico.

$$\begin{array}{r}
 23 \quad 10111 \quad \text{Multiplicando} \\
 19 \quad \times 10011 \quad \text{Multiplicador} \\
 \hline
 10111 \\
 10111 \\
 00000 \quad + \\
 00000 \\
 \hline
 10111 \\
 \hline
 437 \quad 110110101 \quad \text{Producto}
 \end{array}$$

El proceso consiste en buscar primero en los bits sucesivos del multiplicador, el bit menos significativo. Si el bit del multiplicador es un 1, se copia el multiplicando; de otra manera se copian ceros. Los números copiados en líneas sucesivas se recorren una posición a la izquierda a partir del número anterior. Por último, se suman los números y su suma forma el producto.

El signo del producto se determina a partir de los signos del multiplicando y el multiplicador. Si son iguales, el signo del producto es positivo. Si son diferentes, el signo del producto es negativo.

Implantación de hardware para datos de magnitud con signo

Cuando se efectúa la multiplicación en una computadora digital, es conveniente cambiar un poco el proceso. Primero, en lugar de proporcionar registros para almacenar y sumar en forma simultánea la misma cantidad de números binarios que la que hay en el multiplicador, es conveniente proporcionar un sumador para la suma de sólo dos números binarios y acumular sucesivamente los productos parciales en un registro. Segundo, en lugar de ejecutar un corrimiento del multiplicando a la izquierda, el producto parcial se recorre a la derecha, lo que da como resultado la colocación del producto parcial y del multiplicando en las posiciones relativas requeridas. Tercero, cuando el bit correspondiente al multiplicador es 0, no hay necesidad de sumar todos los números 0 al producto parcial porque esto no alterará su valor.

La circuitería para la multiplicación consiste en el equipo que se muestra en la figura 10-1 más dos registros. Estos dos registros junto con los registros A y B se muestran en la figura 10-5. El multiplicador se almacena en el registro Q y su signo en Q_s . El contador secuencial SC se inicializa al principio en un número igual a la cantidad de bits en el multiplicador. El

en A y Q , donde A contiene los bits más significativos y Q contiene los bits menos significativos.

El ejemplo numérico anterior se repite en la tabla 10-2 para hacer más claro el proceso de multiplicación con hardware. El procedimiento sigue los pasos delineados en la tabla de flujo.

TABLA 10-2 Ejemplo numérico para multiplicador binario

Multiplicando $B = 10111$	E	A	Q	SC
Multiplicador en Q	0	00000	10011	101
$Q_n = 1$; sumar B		<u>10111</u>		
Primer producto parcial	0	10111		
Recorrer EAQ a la derecha	0	01011	11001	100
$Q_n = 1$; sumar B		<u>10111</u>		
Segundo producto parcial	1	00010		
Recorrer EAQ a la derecha	0	10001	01100	011
$Q_n = 0$; Recorrer EAQ a la derecha	0	01000	10110	010
$Q_n = 0$; Recorrer EAQ a la derecha	0	00100	01011	001
$Q_n = 1$; sumar B		<u>10111</u>		
Quinto producto parcial	0	11011		
Recorrer EAQ a la derecha	0	01101	10101	000
Producto final en $AQ = 0110110101$				

Algoritmo de multiplicación de Booth

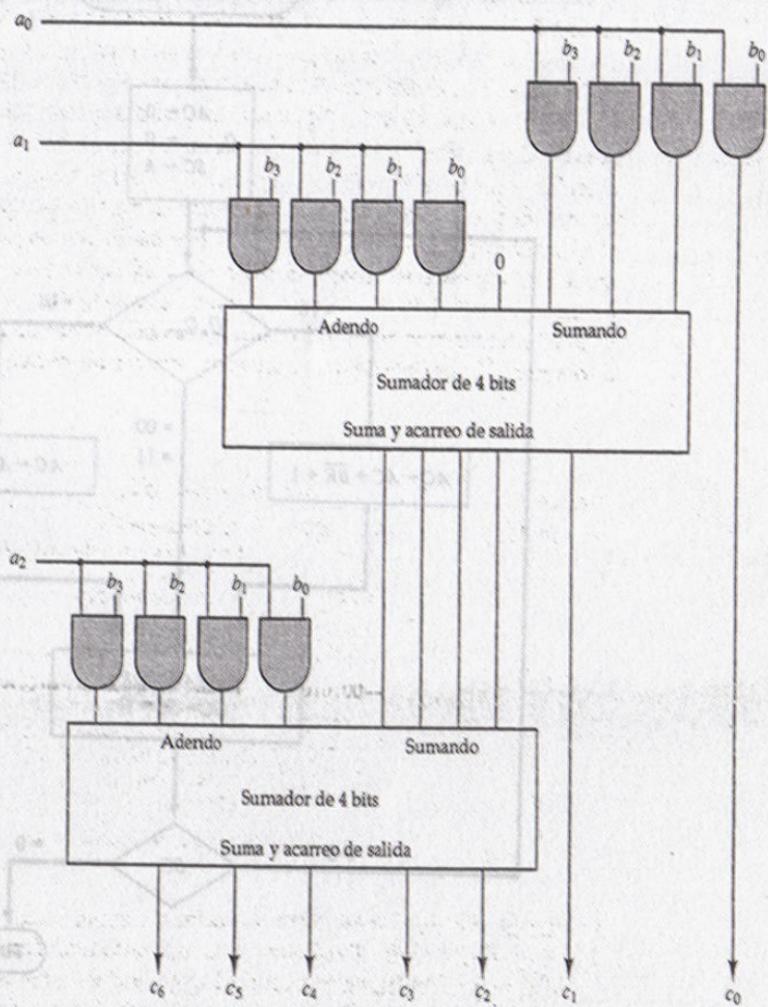
El algoritmo de Booth proporciona un procedimiento para multiplicar enteros binarios en representación de complemento a 2 con signo. Opera con base en que las series de números 0 en el multiplicador no requieren suma, sino sólo corrimiento y que una serie de dígitos 1 en el multiplicador de una ponderación de bit de 2^k a una ponderación 2^m puede tratarse como $2^{k+1} - 2^m$. Por ejemplo, el número binario 001110 (+14) tiene una serie de dígitos 1 de $2^3 + 2^1$ ($k = 3, m = 1$). El número puede representarse como $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$. Por lo tanto, la multiplicación $M \times 14$, donde M es el multiplicando y 14 el multiplicador, puede realizarse como $M \times 2^4 - M \times 2^1$. Como resultado, el producto puede obtenerse al ejecutar el corrimiento del multiplicando binario m cuatro veces a la izquierda y restar una vez M recorrido a la izquierda.

Como en todos los esquemas de multiplicación, el algoritmo de Booth requiere un examen de los bits del multiplicador y ejecutar un corrimiento del producto parcial. Antes del corrimiento, puede sumarse el multiplicando al producto parcial, restarse o dejarlo sin cambio, de acuerdo con las siguientes reglas:

compuertas AND y $(j - 1)k$ sumadores de bits para obtener un producto de $j + k$ bits.

Como segundo ejemplo, consideremos un circuito multiplicador que multiplica un número binario de cuatro bits con un número de tres bits. Representemos el multiplicando mediante $b_3b_2b_1b_0$ y el multiplicador mediante $a_2a_1a_0$. Como $k = 4$ y $j = 3$, necesitamos 12 compuertas AND y dos sumadores de 4 bits para lograr un producto de siete bits. El diagrama lógico del multiplicador se muestra en la figura 10-10.

Figura 10-10. Arreglo multiplicador 4 × 3 bits.



10-4 Algoritmos de división

La división de dos números binarios de punto fijo en representación de magnitud con signo se hace con papel y lápiz mediante un proceso de operaciones de comparar, corrimiento y resta sucesivas. La división binaria es más sencilla que la división decimal porque los dígitos del cociente son 1 o 0 y no hay necesidad de estimar cuántas veces cabe el dividendo del producto parcial dentro del divisor. El proceso de división se ilustra mediante un ejemplo numérico en la figura 10-11. El divisor B consta de cinco bits y el dividendo A de diez bits. Los cinco bits más significativos del dividendo se comparan con el divisor. Como el número de 5 bits es menor que B , probamos de nuevo al tomar los seis bits más significativos de A y comparar este número con B . El número de 6 bits es mayor que B , por lo que colocamos un 1 para el bit del cociente en la sexta posición arriba del dividendo. Después se recorre el divisor una vez a la derecha y se resta del dividendo. La diferencia se denomina *residuo parcial* porque la división podía haberse detenido aquí para obtener un cociente de 1 y un residuo igual al residuo parcial. El proceso continúa al comparar un residuo parcial con el divisor. Si el residuo parcial es mayor o igual que el divisor, el bit del cociente es igual a 1. Después se recorre el divisor a la derecha y se resta del residuo parcial. Si el residuo parcial es menor que el divisor, el bit del cociente es 0 y no se necesita una resta. En cualquier caso el divisor se recorre una vez a la derecha. Nótese que el resultado proporciona un cociente y un residuo.

Divisor: $B = 10001$	11010 0111000000 01110 011100 -10001 ---010110 --10001 ---001010 ---0010100 ----10001 ----000110 -----00110	Cociente = Q Dividendo = A 5 bits de $A < B$, el cociente tiene 5 bits 6 bits de $A \geq B$ Recorrer B a la derecha y restar; introducir 1 en Q 7 bits del residuo $\geq B$ Recorrer B a la derecha y restar; introducir 1 en Q Residuo $< B$; introducir 0 en Q ; recorrer B a la derecha Residuo $\geq B$ Recorrer B a la derecha y restar; introducir 1 en Q Residuo $< B$; introducir 0 en Q Residuo final Divisor
-------------------------	--	---

Figura 10-11 Ejemplo de división binaria.

Implantación de Hardware para datos de magnitud con signo

Cuando se efectúa la división en una computadora digital, es conveniente cambiar el proceso ligeramente. En lugar de ejecutar un corrimiento sobre el divisor a la derecha, el dividendo o residuo parcial se recorre a la izquierda, con lo que se dejan los dos números en la posición relativa

Se dice que un número de punto flotante que tiene un 0 en la posición más significativa de la mantisa, tiene un *sobreflujo inverso*. Para normalizar un número que contiene un sobreflujo inverso, es necesario recorrer la mantisa a la izquierda y decrementar el exponente hasta que aparezca un dígito diferente de cero en la primera posición. En el ejemplo anterior, es necesario recorrer a la derecha dos veces para obtener $.35000 \times 10^3$. En la mayoría de las computadoras, se ejecuta un procedimiento de normalización después de cada operación para asegurar que todos los resultados están en su forma normalizada.

La multiplicación y la división de punto flotante no necesitan un alineamiento de las mantisas. Puede formarse el producto al multiplicar las dos mantisas y sumar los exponentes. La división se realiza al dividir las mantisas y restar los exponentes.

Las operaciones ejecutadas con las mantisas son las mismas que con los números de punto fijo, por lo que los dos tipos pueden compartir los mismos registros y circuitos. Las operaciones ejecutadas con los exponentes son comparar e incrementar (para alinear las mantisas), sumar y restar (para multiplicar y dividir) y decrementar (para normalizar el resultado). El exponente puede representarse en cualquiera de las tres formas siguientes: magnitud de signo, magnitud de complemento a 2 con signo o complemento a 1 con signo.

Un cuarto tipo de representación que se emplea en muchas de las computadoras se conoce como exponente polarizado. En esta representación, se evita que el bit de signo sea una entidad separada. La polarización es un número positivo que se suma a cada exponente mientras se forma el número de punto flotante, para que internamente todos los exponentes sean positivos. El siguiente ejemplo puede hacer más claro este tipo de representaciones. Consideremos un exponente que varía de -50 a 49. Internamente, se representa mediante dos dígitos (sin signo) al sumarle una polarización de 50. El registro del exponente contiene el número $e + 50$, donde e es el exponente real. De esta manera, se representan los exponentes en los registros como números positivos en el rango de 00 a 99. Los exponentes positivos en registros tienen el rango de números de 99 a 50. La resta de 50 proporciona los valores positivos de 49 a 0. Los exponentes negativos se representan en los registros en el rango de 49 a 00. La resta de 50 proporciona los valores negativos en el rango de -1 a -50.

La ventaja de los exponentes polarizados es que sólo contienen números positivos. Por lo tanto, es más sencillo comparar su magnitud relativa sin preocuparse por sus signos. Como consecuencia, puede utilizarse un comparador de magnitud para apreciar la magnitud relativa durante el alineamiento de la mantisa. Otra ventaja es que el exponente polarizado más pequeño posible contiene sólo números 0. De esa manera la representación de punto flotante de 0 es una mantisa 0 y el exponente más pequeño posible.

En los ejemplos anteriores, utilizamos números decimales para mostrar algunos de los conceptos que deben comprenderse cuando se trabaja con

números de punto flotante. Es obvio que los mismos conceptos se aplican también a los números binarios. Los algoritmos desarrollados en esta sección son para números binarios. La aritmética decimal de computadora se analiza en la sección siguiente.

Configuración de registros

La configuración de registros para operaciones de punto flotante es muy similar a la descrita para operaciones de punto fijo. Como regla general, para procesar las mantisas se usan los mismos registros y el mismo sumador para las operaciones de punto fijo. La diferencia estriba en la manera en que se manejan los exponentes.

La organización de registros para operaciones de punto flotante se muestra en la figura 10-14. Existen tres registros, BR, AC y QR. Cada registro se subdivide en dos partes. La parte de la mantisa tiene los mismos símbolos de letras mayúsculas que la representación de punto fijo. La parte del exponente utiliza el símbolo de letra minúscula correspondiente.

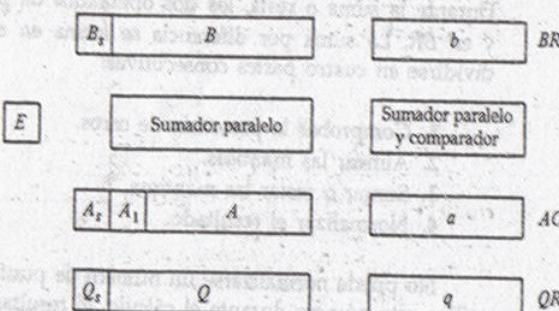


Figura 10-14 Registros para operaciones aritméticas de punto flotante.

Se considera que cada número de punto flotante tiene una mantisa en representación de magnitud con signo y un exponente polarizado. Por lo tanto, el AC tiene una mantisa cuyo signo está en A_s y una magnitud que está en A . El exponente está en la parte del registro representada por el símbolo de la letra a minúscula. El diagrama muestra en forma explícita el bit más significativo de A etiquetado mediante A_1 . El bit en esta posición debe ser un 1 para el número que va a normalizarse. Nótese que el símbolo AC representa todo el registro, esto es, la concatenación de A_s , A y a .

De igual manera, el registro BR se subdivide en B_s , B y b , y QR en Q_s , Q y q . Un sumador paralelo suma las dos mantisas y transfiere el resultado a A y el acarreo a E . Para los exponentes se utiliza un sumador paralelo separado. Como los exponentes están polarizados, no tienen un bit de signo

distinto, pero se representan como una cantidad positiva polarizada. Se considera que los números de punto flotante son tan grandes que es muy remota la posibilidad de un sobreflujo de exponente, y por esta razón, no se considera un sobreflujo de exponentes. Los exponentes también se conectan a un comparador de magnitud que proporciona tres salidas binarias para indicar su magnitud relativa.

El número de la mantisa se tomará como una fracción, para que se considere que el punto binario reside a la izquierda de la parte de magnitud. La representación entera para punto flotante produce ciertos problemas en el establecimiento de escalas, durante la multiplicación y la división. Para evitar estos problemas, adoptamos una representación de fracciones.

Se considera que, inicialmente, los números de los registros están normalizados. Después de cada operación aritmética, se normalizará el resultado. Conforme a eso, todos los operandos de punto flotante que se mueven de y a la unidad de memoria están siempre normalizados.

Suma y resta

Durante la suma o resta, los dos operandos de punto flotante están en AC y en BR. La suma por diferencia se forma en el AC. El algoritmo puede dividirse en cuatro partes consecutivas:

1. Comprobar la presencia de ceros.
2. Alinear las mantisas.
3. Sumar o restar las mantisas.
4. Normalizar el resultado.

No puede normalizarse un número de punto flotante que es cero. Si se utiliza este número durante el cálculo, el resultado también puede ser cero. En lugar de comprobar la presencia de ceros durante el proceso de normalización, comprobamos los ceros en un principio y, si es necesario, se termina el proceso. Debe realizarse el alineamiento de las mantisas antes de su operación. Después de que se suman o restan las mantisas, el resultado puede estar sin normalizar. El procedimiento de normalización asegura que el resultado esté normalizado antes de su transferencia a memoria.

El diagrama de flujo para sumar o restar dos números binarios de punto flotante se muestra en la figura 10-15. Si BR es igual a cero, se termina operación, y el valor en AC es el resultado. Si AC es igual a cero, transferimos el contenido de BR a AC y, si se van a restar los números, también complementamos su signo. Si ninguno de los números es igual a cero, procedemos a alinear las mantisas.

El comparador de magnitud conectado a los exponentes a y b proporciona tres salidas que indican su magnitud relativa. Si los dos exponentes son iguales, pasamos a ejecutar la operación aritmética. Si los exponentes son diferentes, la mantisa con el exponente menor se desplaza a la derecha y se

en 0 y la operación se termina. Si ninguno es igual a 0 el proceso continúa con la suma de exponentes.

El exponente del multiplicador está en q y el sumandor está entre los exponentes a y b . Es necesario transferir los exponentes de q a a , sumar los dos exponentes y transferir la suma a a . Como ambos exponentes se polarizan

Figura 10-16 Multiplicación de números de punto flotante.

Multiplicar

Multiplicando en BR
Multiplicador en QR

$BR = 0$

$QR \neq 0$

$AC \leftarrow 0$

$a \leftarrow q$

$a \leftarrow a + b$

$a \leftarrow a - \text{bias}$

Multiplicar mantisa
como en fig. 10-6

$shl\ A_Q$
 $a \leftarrow a - 1$

$A_1 = 0$

$A_1 = 1$

FIN
(producto en AC)

3. Alinear el dividendo.
4. Restar los exponentes.
5. Dividir las mantisas.

En la figura 10-17 se muestra el diagrama de flujo para la división de punto flotante. Se comprueban los dos operandos para saber si hay ceros. Si el divisor es cero, indica un intento por dividir entre cero, que es una operación ilegal. Se termina la operación con un mensaje de error. Un procedimiento alternativo sería establecer el cociente de QR en el número más positivo posible (si el dividendo es positivo). O el número más negativo posible (si el dividendo es negativo). Si el dividendo en AC es cero, el cociente en QR se hace cero y termina la operación.

Si los operandos no son cero, avanzamos para determinar el signo del cociente y almacenarlo en Q_s . El signo del dividendo en A_s no se afecta para que sea el signo del residuo. El registro Q se borra y el contador secuencial SC se inicializa en un número igual a la cantidad de bits en el cociente.

El alineamiento del dividendo es similar a la prueba del sobreflujo al dividir en la operación de punto fijo. El alineamiento adecuado requiere que el dividendo de la fracción sea menor que el divisor. Se comparan las dos fracciones mediante una prueba de resta. El acarreo en E determina su magnitud relativa. Se restablece la fracción del dividendo a su valor original al sumar el divisor. Si $A \geq B$, es necesario recorrer a la derecha A una vez e incrementar el exponente del dividendo. Como ambos operandos están normalizados, este alineamiento asegura que $A < B$.

Después, se resta el exponente divisor del exponente dividendo. Como ambos exponentes estaban polarizados originalmente, la operación de resta proporciona la diferencia sin el polo. Después se suma el polo y el resultado se transfiere a q , porque el cociente se formó en QR.

Se divide la magnitud de las mantisas como en el caso del punto flotante. Después de la operación, el cociente mantisa reside en Q y el residuo en A . El cociente de punto flotante ya está normalizado y reside en QR. El exponente del residuo debe ser igual al exponente del dividendo. El punto binario para la mantisa residuo se encuentra a $(n - 1)$ posiciones a la izquierda de A_1 . Puede convertirse el residuo en una fracción normalizada al restar $n - 1$ del exponente del dividendo y al recorrer y decrementar hasta que el bit en A_1 sea igual a 1. Esto no se muestra en el diagrama de flujo y se deja como ejercicio.

10-6 Unidad aritmética decimal

El usuario de una computadora prepara datos con números decimales y recibe resultados en forma decimal. Una CPU con una unidad aritmética-lógica puede ejecutar microoperaciones aritméticas con datos binarios. Para ejecutar operaciones aritméticas con datos decimales, es necesario convertir

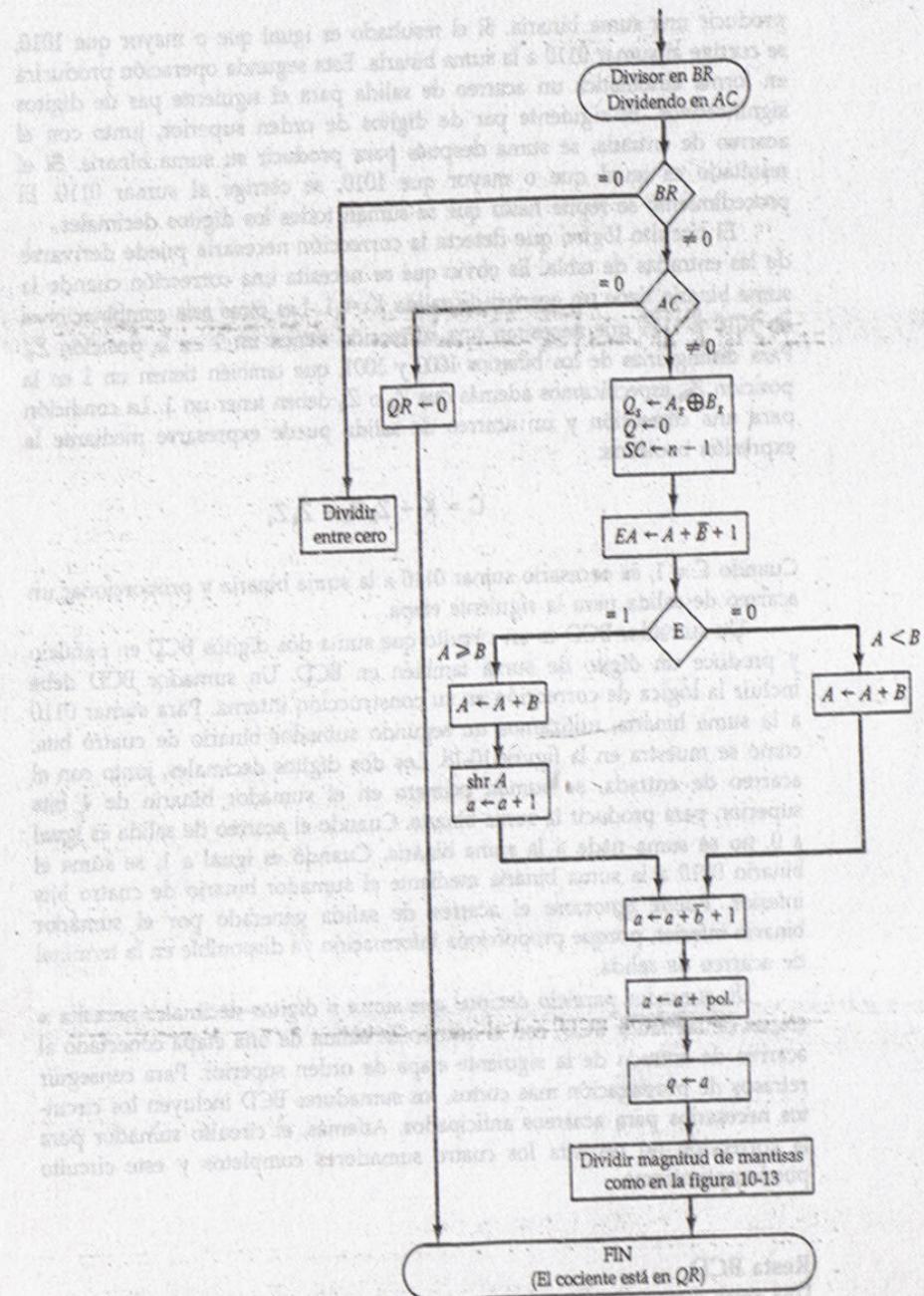


Figura 10-17 División de números de punto flotante.

los números decimales de entrada en binarios, para ejecutar todos los cálculos con números binarios y convertir el resultado en decimal. Este puede ser un método eficiente en aplicaciones que requieran una gran cantidad de cálculos y una cantidad relativamente más pequeña de datos de entrada y salida. Cuando la aplicación solicita una gran cantidad de entradas y salidas y un número relativamente más pequeño de cálculos aritméticos, es más conveniente hacer la aritmética interna de manera directa con los números decimales. Las computadoras capaces de ejecutar aritmética decimal deben almacenar los datos decimales en forma de código binario. Despues se aplican los números decimales a una unidad aritmética decimal capaz de ejecutar microoperaciones aritméticas decimales.

Las calculadoras electrónicas utilizan invariabilmente una unidad aritmética decimal interna, debido a que las entradas y salidas son frecuentes. No parece existir una razón para convertir los números de entrada del teclado en binario y volver a convertir los resultados exhibidos en decimales porque este proceso requiere circuitos especiales y también necesita más tiempo para ejecutarse. Muchas computadoras tienen hardware para cálculos aritméticos con datos binarios y decimales. Los usuarios pueden especificar mediante instrucciones programadas si desean que la computadora ejecute los cálculos con datos binarios o decimales.

Una unidad aritmética decimal es una función digital que ejecuta microoperaciones decimales. Puede sumar o restar números decimales, por lo regular, al formar el complemento a 9 o 10 del sustraendo. La unidad acepta números decimales codificados y genera resultados en el mismo código binario adoptado. Una unidad aritmética decimal de una sola etapa consta de nueve variables binarias de entrada y cinco variables binarias de salida, porque se necesita un mínimo de cuatro bits para representar cada dígito decimal codificado. Cada etapa debe tener cuatro entradas para un dígito sumando, cuatro entradas para el otro sumando y un acarreo de entrada. Las salidas incluyen cuatro terminales para el dígito de suma y uno para el acarreo de salida. Por supuesto, existe una amplia variedad de configuraciones de circuito posibles, que dependen del código utilizado para representar los dígitos decimales.

Sumador BCD

Consideremos la suma aritmética de dos dígitos decimales en BCD, junto con un posible acarreo de una etapa anterior. Como cada dígito de entrada no excede de 9, la suma de salida no puede ser mayor que $9 + 9 + 1 = 19$, el 1 en la suma es un acarreo de entrada. Supongamos que se aplican dos dígitos BCD a un sumador binario de 4 bits. El sumador formará la suma en binario y producirá un resultado que puede variar de 0 a 19. Estos números binarios se listan en la tabla 10-4 y se etiquetan mediante los símbolos K , Z_8 , Z_4 , Z_2 y Z_1 . K es el acarreo y los subíndices bajo la letra Z representan las ponderaciones 8, 4, 2 y 1 que pueden asignarse a los cuatro

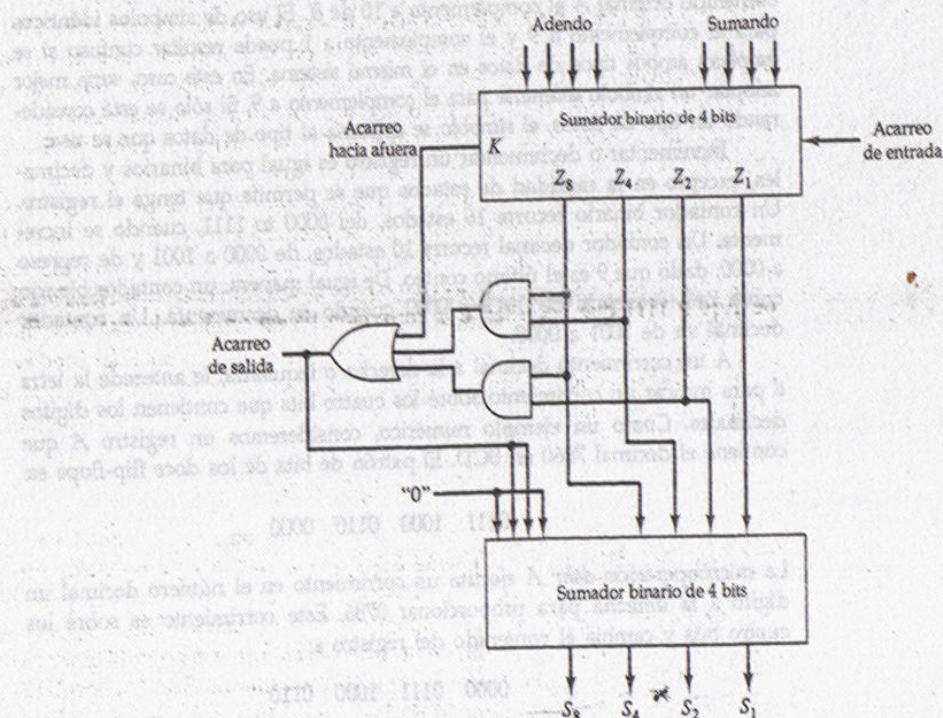


Figura 10-18 Diagrama de bloques de un sumador BCD.

minuendo. Como el BCD no es un código autocomplementario, no puede obtenerse el complemento a 9 al complementar cada bit en el código. Debe formarse mediante un circuito que reste cada dígito BCD de 9.

El complemento a 9 de un dígito decimal representado en BCD puede obtenerse al complementar los bits en la representación codificada del dígito, siempre y cuando se incluya una corrección. Existen dos métodos de corrección posibles. En el primero, se suma el binario 1010 (decimal 10) a cada dígito complementado y al acarreo descartado después de cada suma. En el segundo método, se suma el número binario 0110 (decimal 6) antes de que se complemente el dígito. Como un ejemplo numérico, el complemento a 9 del BCD 0111 (decimal 7) se calcula al complementar primero cada bit para obtener 1000. Al sumar el binario 1010 y descartar el acarreo, obtenemos 0010 (decimal 2). Mediante el segundo método sumamos 0110 a 0111 para obtener 1101. Al complementar cada bit, obtenemos el resultado requerido de 0010. Complementar cada bit de un número binario N de cuatro bits es igual a restar el número de 1111 (decimal 15). Sumar el equivalente binario del decimal 10 proporciona $15 - N + 10 = 9 - N + 16$. Pero 16 significa el

Una etapa de una unidad aritmética decimal que puede sumar o restar dos dígitos BCD se muestra en la figura 10-19. Consiste en un sumador BCD y un complementador a 9. El modo M controla la operación de la unidad. Con $M = 0$ las salidas S forman la suma de A y B . Con $M = 1$, las salidas S forman la suma de A más el complemento a 9 de B . Para números con n dígitos decimales necesitamos n etapas como ésa. El acarreo de salida C_{n+1} de una etapa debe conectarse al acarreo de entrada C_i de la siguiente etapa de orden superior. La mejor manera de restar los dos números decimales es dejar $M = 1$ y aplicar un 1 al acarreo de entrada C_1 de la primera etapa. Las salidas formarán la suma de A más el complemento a 10 de B , lo cual es equivalente a una operación de resta si se descarta el acarreo de la última etapa.

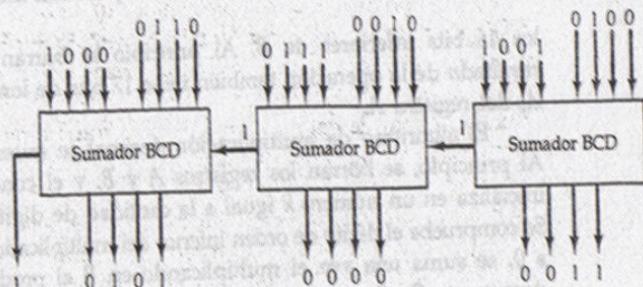
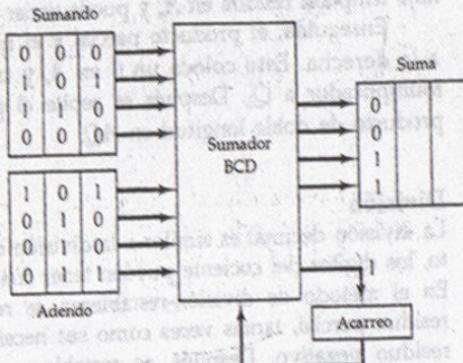
10-7 Operaciones aritméticas decimales

Los algoritmos para las operaciones aritméticas con datos decimales son similares a los algoritmos para las operaciones correspondientes de datos binarios. De hecho, excepto por una ligera modificación en los algoritmos de multiplicación y división, pueden usarse los mismos diagramas de flujo para ambos tipos de datos, siempre y cuando interpretemos los símbolos de la microoperación en forma adecuada. Los números decimales en BCD se almacenan en los registros de la computadora en grupos de cuatro bits. Cada grupo de cuatro bits representa un dígito decimal y debe tomarse como una unidad cuando se ejecutan microoperaciones decimales.

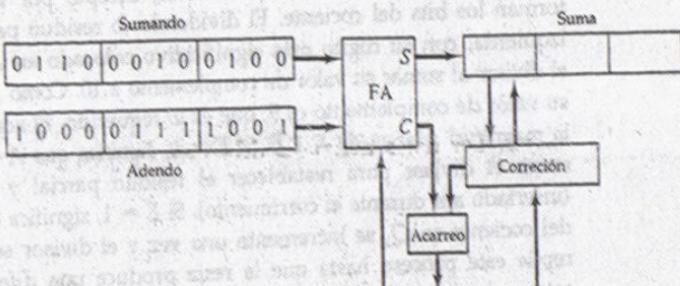
Por conveniencia, utilizaremos los mismos símbolos para microoperaciones aritméticas binarias y decimales, pero les daremos una interpretación diferente. Como se muestra en la tabla 10-5 una barra sobre el símbolo de la letra del registro representa el complemento a 9 del número decimal almacenado en el registro. Al sumar 1 al complemento a 9 se produce el complemento a 10. Por lo tanto, para números decimales, el símbolo $A \leftarrow A + B + 1$ representa una transferencia de la suma decimal formada al sumar el

TABLA 10-5 Símbolos de microoperaciones aritméticas decimales

Representación simbólica	Descripción
$A \leftarrow A + B$	Sumar números decimales y transferir la suma dentro de A
\bar{B}	Complemento a dígitos 9 de B
$Q_L \leftarrow Q_L + 1$	Contenido de A más complemento a dígitos 10 de B dentro de A
$A \leftarrow A + \bar{B} + 1$	Incrementar número BCD en Q_L
dshr A	Corrimiento decimal a la derecha del registro A
dshl A	Corrimiento decimal a la izquierda del registro A

a) Suma decimal paralela: $624 + 879 = 1503$ 

b) Suma decimal de dígito serial, bit paralelo



c) Suma decimal por completo serial

Figura 10-20 Tres maneras de sumar números decimales

los 16 bits inferiores de B . Al principio se borran tanto B_e como A_e . El resultado de la operación también tiene 17 bits de longitud y no usa la parte A_e del registro A .

El algoritmo de multiplicación decimal se muestra en la figura 10-22. Al principio, se borran los registros A y B_e , y el contador secuencial SC se inicializa en un número k igual a la cantidad de dígitos en el multiplicador. Se comprueba el dígito de orden inferior del multiplicador en Q_L . Si es diferente a 0, se suma una vez el multiplicando en B al producto parcial en A y se decremente Q_L . Se comprueba Q_L una vez más y se repite el proceso hasta que es igual a 0. De esta manera, el multiplicando en B se suma al producto parcial una cantidad de veces igual al dígito multiplicador. Cualquier dígito de sobreflujo temporal residirá en A_e y puede variar su valor de 0 a 9.

Enseguida, el producto parcial y el multiplicador se recorren una vez a la derecha. Esto coloca un 0 en A_e y transfiere el siguiente dígito del multiplicador a Q_L . Después se repite el proceso k veces para formar un producto de doble longitud en AQ .

División

La división decimal es similar a la división binaria, excepto que, por supuesto, los dígitos del cociente pueden tener cualquiera de diez valores de 0 a 9. En el método de división-restablecer, se resta el divisor del dividendo o residuo parcial, tantas veces como sea necesario, hasta que se produzca un residuo negativo. Después, se restablece el residuo correcto al sumar el divisor. El dígito en el cociente refleja la cantidad de restas acumuladas, pero excluye la que produjo la diferencia negativa.

El algoritmo de división decimal se muestra en la figura 10-23. Es similar al algoritmo con datos binarios, excepto por la manera en que se forman los bits del cociente. El dividendo (o residuo parcial) se recorre a la izquierda, con su dígito más significativo colocado en A_e . Después, se resta el divisor al sumar su valor de complemento a 10. Como B_e se limpió al inicio, su valor de complemento es 9, que es lo requerido. El acarreo en E determina la magnitud relativa de A y B . Si $E = 0$, significa que $A < B$. En este caso, se suma el divisor para restablecer el residuo parcial y Q_L se queda en 0, (insertado ahí durante el corrimiento). Si $E = 1$, significa que $A \geq B$. El dígito del cociente en Q_L se incrementa una vez y el divisor se resta de nuevo. Se repite este proceso hasta que la resta produce una diferencia negativa que se reconoce cuando E es 0. Cuando ocurre esto, el dígito del cociente no se incrementa, pero se suma el divisor para restablecer el residuo positivo. De esta manera, el dígito del cociente se hace igual a la cantidad de veces que el residuo parcial "va" dentro del divisor.

Los bits del residuo parcial y del cociente se recorren una vez a la izquierda y el proceso se repite k veces para formar k dígitos de cociente. Después se encuentra el residuo en el registro A y el cociente está en el registro Q . No se considera el valor de E .

EN ESTE CAPÍTULO

- 11-1** Dispositivos periféricos
- 11-2** Interface de entrada-salida
- 11-3** Transferencia asíncrona de datos
- 11-4** Modos de transferencia
- 11-5** Prioridad de interrupción
- 11-6** Acceso directo a memoria (DMA)
- 11-7** Procesador de entrada-salida (IOP)
- 11-8** Comunicación serial

11-1 Dispositivos periféricos

E/S

El subsistema de entrada-salida de una computadora, denominado E/S, proporciona un modo de comunicación eficiente entre el sistema central y el ambiente externo. Los programas y datos deben introducirse a la memoria de la computadora para su procesamiento y los resultados que se obtienen de los cálculos deben grabarse o registrarse para el usuario. Una computadora no tiene ningún propósito útil sin la capacidad de recibir información de una fuente externa y de transmitir los resultados de manera comprensible.

El medio más familiar de introducir información en una computadora es a través de un teclado tipo máquina de escribir, que permite a una persona introducir información alfanumérica en forma directa. Cada vez que se oprime una tecla, la terminal envía un carácter codificado en binario a la computadora. La velocidad más alta posible para introducir información de esta manera depende de la velocidad para teclear de una persona. Por otra parte, la unidad de procesamiento central es un dispositivo extremadamente rápido capaz de ejecutar operaciones a muy alta velocidad. Cuando se transfiere a un procesador información de entrada mediante un teclado lento,

que existen entre la computadora central y cada periférico. Las diferencias principales son:

1. Los periféricos son dispositivos electromecánicos y electromagnéticos y su manera de operación es diferente a la de la CPU y la memoria, que son dispositivos electrónicos. Por lo tanto, puede requerirse una conversión de valores de señales.
2. La velocidad de transferencia de datos de los periféricos, por lo general, es menor que la velocidad de transferencia de la CPU y, en consecuencia, puede necesitarse un mecanismo de sincronización.
3. Los códigos de datos y los formatos en los periféricos son diferentes del formato de la palabra en la CPU y en la memoria.
4. Los modos de operación de los periféricos son diferentes uno de otro y cada uno debe estar controlado para no perturbar la operación de otros periféricos conectados a la CPU.

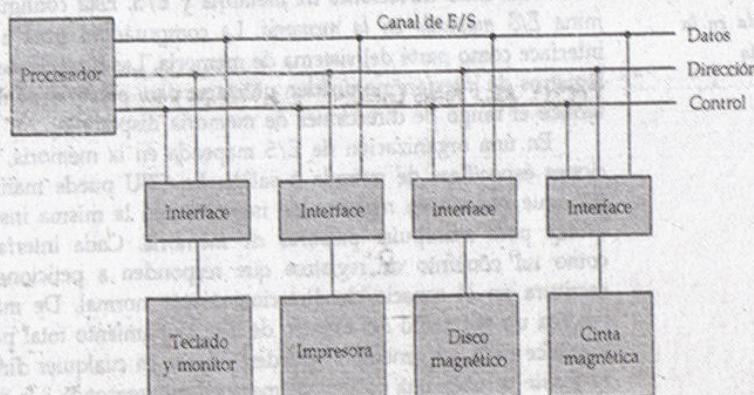
Para resolver estas diferencias, los sistemas de computadora incluyen componentes de circuitería especiales entre la CPU y los periféricos para supervisar y sincronizar todas las transferencias de entrada y salida. Estos componentes se llaman *interfaces*, porque se comunican tanto con el canal del procesador como con el dispositivo periférico.

Además, cada dispositivo puede tener su propio controlador que supervisa las operaciones del mecanismo particular en el periférico.

Canal de E/S y módulos de interface

Un enlace de comunicación típico entre el procesador y varios periféricos se muestra en la figura 11-1. El canal de E/S, consta de líneas de datos, líneas

Figura 11-1 Conexión de canal de E/S a dispositivos de entrada-salida.



un comando de control. Después, el procesador monitorea el estado de la cinta mediante un comando de estado. Cuando la cinta está en la posición correcta, el procesador envía un comando de salida de datos. La interface responde a la dirección y al comando y transfiere la información de las líneas de datos del canal a su registro intermedio (buffer). En seguida, la interface comunica con el controlador de la cinta y envía los datos que se van a almacenar.

datos de entrada

El comando de entrada de datos es lo opuesto al de salida de datos. En este caso, la interface recibe datos del periférico y los coloca en su registro intermedio. El procesador verifica si los datos están disponibles mediante un comando de estado y después envía un comando de entrada de datos. La interface coloca los datos sobre las líneas de datos, donde el procesador los acepta.

E/S versus canal de memoria

Además de comunicarse con su espacio de E/S, el procesador debe comunicarse con la unidad de memoria. Como el canal de E/S, el canal de memoria contiene datos, direcciones y líneas de control de lectura/escritura. Existen tres maneras que pueden utilizar los canales de la computadora para comunicarse con la memoria y las E/S:

1. Utilizar dos canales separados, uno para la memoria y el otro para las E/S.
2. Utilizar un canal común para memoria y E/S pero tener líneas de control separadas para cada una.
3. Utilizar un canal común para memoria y E/S con líneas de control comunes.

En el primer método, la computadora tiene conjuntos de canales de datos, de control y de direcciones independientes, uno para accesar la memoria y el otro para las E/S. Esto se hace en computadoras que proporcionan un procesador de E/S separado (*I/O processor, IOP*) además de la unidad de procesamiento central (CPU). La memoria se comunica con la CPU y el IOP por medio de un canal de memoria. El IOP se comunica también con los dispositivos de entrada y salida mediante un canal de E/S separado, con sus direcciones, datos y líneas de control. El propósito del IOP es proporcionar una trayectoria independiente para la transferencia de información entre dispositivos externos y la memoria interna. El procesador de E/S en algunas ocasiones se denomina *canal de datos*. En la sección 11-7 analizamos con mayor detalle la función del IOP.

E/S aislada versus E/S mapeada en memoria

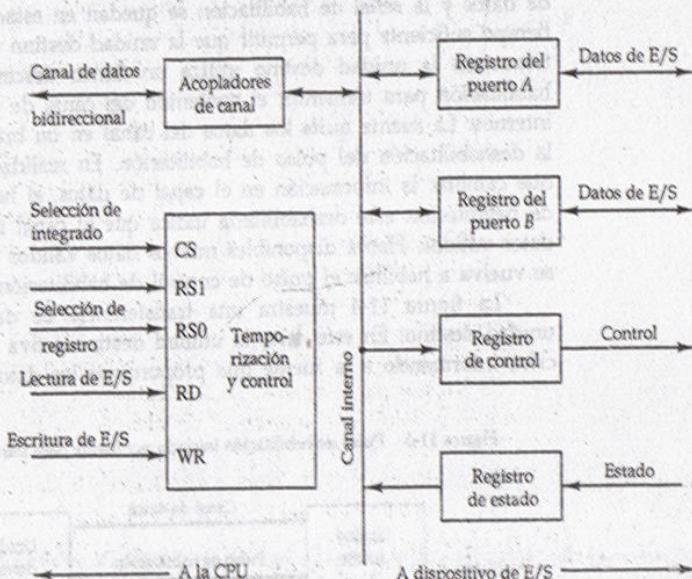
Muchas computadoras utilizan un canal común para transferir información entre la memoria o la E/S y la CPU. La diferencia entre una transferencia

computadora utilizar las mismas instrucciones para transferencia de entrada-salida o para transferencia de memoria. La ventaja es que las instrucciones de carga y almacenamiento utilizadas para leer y escribir en la memoria pueden utilizarse para introducir y sacar datos de los registros de E/S. En una computadora típica, hay más instrucciones de referencia a memoria que instrucciones de E/S. Con las E/S mapeadas en memoria todas las instrucciones que hacen referencia a memoria también están disponibles para E/S.

Ejemplo de interface E/S

Un ejemplo de una interface de E/S se muestra en forma de diagrama de bloques en la figura 11-2. Consta de dos registros de datos llamados *puertos*,

Figura 11-2 Ejemplo de unidad de interface de E/S.



CS	RS1	RS0	Registro seleccionado
0	x	x	Ninguno: canal de datos en alta impedancia
1	0	0	Registro del puerto A
1	0	1	Registro del puerto B
1	1	0	Registro de control
1	1	1	Registro de estado

un registro de control, un registro de estado, acopladores de canal y circuitos de temporización de control. La interface se comunica con la CPU mediante el canal de datos. Las entradas de selección de integrado y de selección de registro determinan la dirección asignada a la interface. Lectura de E/S y escritura de E/S son dos líneas de control que especifican una entrada o salida, respectivamente. Los cuatro registros comunican en forma directa con un dispositivo de E/S conectado a la interface. Los datos de E/S hacia y desde el dispositivo pueden transferirse al puerto A o al puerto B. La interface puede operar con un dispositivo de salida, con un dispositivo de entrada o con un dispositivo que requiere tanto entradas como salidas. Si la interface está conectada a una impresora, sólo sacará datos, y si da servicio a un lector de caracteres, sólo introducirá datos. Una unidad de disco magnético transfiere datos en ambas direcciones pero no al mismo tiempo, por lo que la interface puede utilizar líneas bidireccionales. Se pasa un comando al dispositivo de E/S al enviar una palabra al registro apropiado de la interface. En un sistema como este, no se necesita el código de función en el canal de E/S, porque el comando se envía al registro de control, la información de estado se recibe del registro de estado y los datos se transfieren hacia los registros de los puertos A y B. Por lo tanto, la transferencia de datos, el control y la información de estado se realizan siempre mediante el canal de datos común. La diferencia entre datos, control o información de estado se determina del registro particular de la interface con el que se comunica la CPU.

El registro de control recibe información de control de la CPU. Al cargar los bits apropiados dentro del registro de control, la interface y el dispositivo de E/S conectados a ella pueden colocarse en diversos modos de operación. Por ejemplo, el puerto A puede definirse como un puerto de entrada y el puerto B como un puerto de salida. Pueden darse instrucciones a una unidad de cinta magnética para que rebobine la cinta o para que se arranque con un movimiento hacia adelante. Los bits en el registro de estado se utilizan para condiciones de estado y para registrar errores que pueden ocurrir durante la transferencia de datos. Por ejemplo, un bit de estado puede indicar que el puerto A ha recibido un nuevo conjunto de datos del dispositivo de E/S. Otro bit del registro de estado puede indicar qué ha ocurrido un error de paridad durante la transferencia.

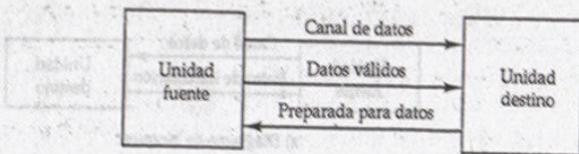
Los registros de la interface comunican con la CPU por medio del canal de datos bidireccional. El canal de direcciones selecciona la unidad de interface por medio de la entrada de selección de integrado y de las dos entradas de selección de registros. En forma externa, debe proporcionarse un circuito (por lo general un decodificador) para detectar la dirección asignada a los registros de la interface. El circuito habilita la entrada de selección de integrado (*chip select, CS*) cuando se selecciona la interface mediante el canal de direcciones. Las dos entradas de selección de registro RS1 y RS0, por lo general, se conectan a las dos líneas menos significativas del canal de direcciones. Estas dos entradas seleccionan uno de los cuatro registros en la

destino que inicie la transferencia no puede saber si la unidad fuente ha colocado realmente los datos en el canal. El método de reconocimiento mutuo (handshake) resuelve este problema al introducir una segunda señal de control que proporciona una respuesta a la unidad que inicia la transferencia. El principio básico del método de reconocimiento mutuo de dos líneas de transferencia de datos es el siguiente. Una línea de control está en la misma dirección que el flujo de datos en el canal, de la fuente al destino. La utiliza la unidad fuente para informar a la unidad destino si hay datos válidos en el canal. La otra línea de control está en la dirección opuesta. La utiliza la unidad destino para informar a la fuente si puede aceptar datos. La secuencia de control durante la transferencia depende de la unidad que inicia la transferencia.

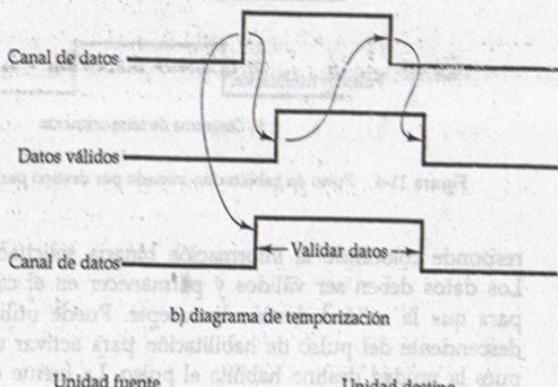
La figura 11-5 muestra el procedimiento de transferencia de datos cuando lo inicia la fuente. Las dos líneas de reconocimiento mutuo son *datos válidos*, que genera la unidad fuente y, *datos aceptados*, generada por la unidad destino. El diagrama de temporización muestra el intercambio de señales entre las dos unidades. La secuencia de eventos listada en la parte (c) muestra los cuatro estados posibles que puede tener el sistema en cualquier momento. La unidad fuente inicia la transferencia al colocar los datos en el canal y habilitar su señal de *datos válidos*. La unidad destino activa la señal de *datos aceptados* después de que acepta los datos del canal. En seguida, la unidad fuente deshabilita su señal de *datos válidos*, la cual invalida los datos en el canal, después la unidad destino deshabilita su señal de *datos aceptados* y el sistema pasa a su estado inicial. La unidad fuente no envía los datos siguientes hasta después que la unidad destino muestra su disponibilidad para aceptar nuevos datos al deshabilitar su señal de *datos aceptados*. Este esquema permite retrasos arbitrarios de un estado al siguiente y que cada unidad responda a su propia velocidad de transferencia de datos. La velocidad de transferencia está determinada por la unidad más lenta.

La transferencia que utiliza líneas de reconocimiento mutuo iniciada por la unidad destino se muestra en la figura 11-6. Nótese que el nombre de la señal generada por la unidad destino se ha cambiado a *preparada para datos*, con el fin de que refleje su nuevo significado. En este caso, la unidad fuente no coloca datos en el canal hasta que recibe la señal *preparada para datos* de la unidad destino. De ahí en adelante, el procedimiento de reconocimiento mutuo sigue el mismo patrón que en el caso iniciado por la unidad fuente. Nótese que la secuencia de eventos en ambos casos sería idéntica si consideramos la señal de *preparada para datos* como el complemento de *datos aceptados*. De hecho, la única diferencia entre la transferencia iniciada por la fuente y la iniciada por el destino, es en la elección de su estado inicial.

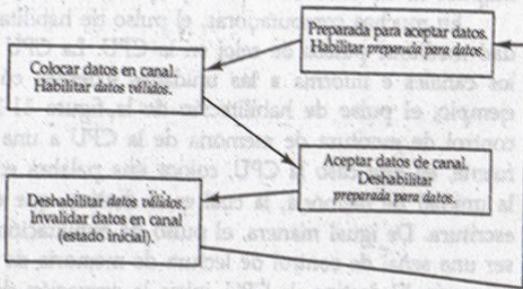
El esquema de reconocimiento mutuo proporciona un alto grado de flexibilidad y confiabilidad porque el término exitoso de una transferencia de datos se basa en la participación activa de ambas unidades. Si una unidad está defectuosa, no se completará la transferencia de datos. El error puede detectarse mediante un mecanismo de *tiempo transcurrido*, que produce una



a) Diagrama de bloque



b) Diagrama de temporización



c) Secuencia de eventos

Deshabilitar datos válidos.

Figura 11-6 Transferencia iniciada por destino utilizando reconocimiento mutuo.

Transferencia serial asíncrona

La transferencia de datos entre dos unidades puede hacerse en forma paralela o serial. En la transmisión de datos paralela, cada bit en el mensaje tiene su propia trayectoria y todo el mensaje se transmite al mismo tiempo. Esto significa que un mensaje de n bits debe transmitirse a través de n trayectorias conductoras separadas. En la transferencia de datos serial, cada bit en el mensaje se envía en secuencia uno a la vez. Este método requiere el uso de un par de conductores o un conductor y una

bit de paro

Al usar estas reglas, el receptor puede detectar el bit de inicio cuando la línea pasa de 1 a 0. Un reloj en el receptor examina la línea en los tiempos de bit convenientes. El receptor conoce la velocidad de transferencia de los bits y la cantidad de bits de caracteres que debe aceptar. Después de que se transmiten los bits de caracteres, se envían uno o dos bits de paro. Los bits de paro están siempre en el estado 1 y marcan el fin del carácter para dar a entender el estado desocupado o de espera.

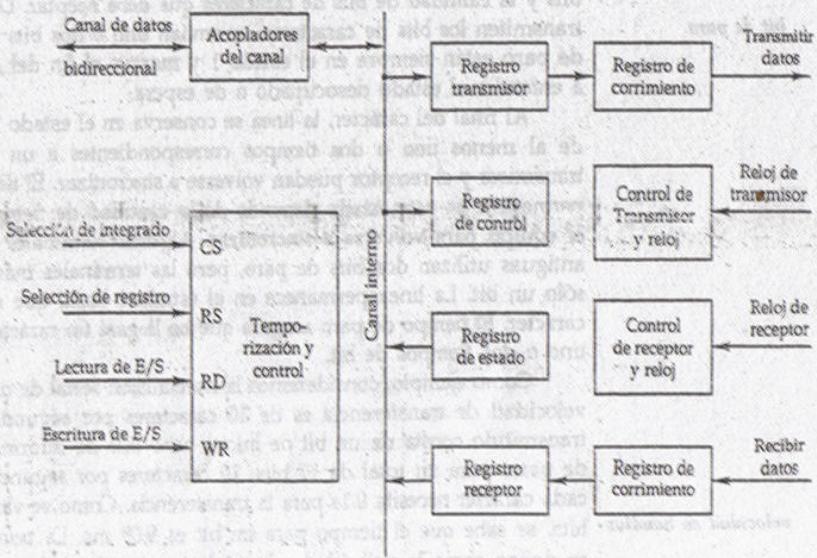
Al final del carácter, la línea se conserva en el estado 1 por un período de al menos uno o dos tiempos correspondientes a un bit, para que el transmisor y el receptor puedan volverse a sincronizar. El tiempo que la línea permanece en este estado depende de la cantidad de tiempo requerido por el equipo para volverse a sincronizar. Algunas terminales electromecánicas antiguas utilizan dos bits de paro, pero las terminales más nuevas utilizan sólo un bit. La línea permanece en el estado 1 hasta que se transmite otro carácter. El tiempo de paro asegura que no llegará un carácter nuevo durante uno o dos tiempos de bit.

velocidad en baudios

Como ejemplo, consideremos la transmisión serial de una terminal cuya velocidad de transferencia es de 10 caracteres por segundo. Cada carácter transmitido consta de un bit de inicio, ocho bits de información, y dos bits de paro, para un total de 11 bits. 10 caracteres por segundo significan que cada carácter necesita 0.1s para la transferencia. Como se van a transmitir 11 bits, se sabe que el tiempo para un bit es 9.09 ms. La *velocidad en baudios* se define como la velocidad a la cual se transmite información serial y es equivalente a la transferencia de datos en bits por segundo. Diez caracteres por segundo con un formato de 11 bits tiene una velocidad de transferencia de 110 baudios. La terminal tiene un teclado y una impresora. Cada vez que se oprime una tecla, una terminal envía 11 bits en forma serial a lo largo de una línea. Para imprimir un carácter en la impresora, debe recibirse un mensaje de 11 bits a través de otra línea. La interface de la terminal consta de un transmisor y un receptor. El transmisor acepta un carácter de 8 bits de la computadora y procede a enviar un mensaje serial de 11 bits a través de la línea de la impresora. El receptor acepta el mensaje serial de 11 bits de la línea del teclado e introduce el código de carácter de 8 bits dentro de la computadora. Están disponibles circuitos integrados diseñados en forma específica para proporcionar la interface entre la computadora y terminales interactivas similares. Tal circuito se denomina una *interface de comunicación asíncrona o receptor-transmisor asíncrono universal (universal asynchronous receiver-transmitter, UART)*.

Interface de comunicación asíncrona

El diagrama de bloque de una interface de comunicación asíncrona se muestra en la figura 11-8. Funciona como transmisor y receptor. La interface se inicializa para un modo de transferencia particular mediante un byte de control que se carga dentro de su registro de control. El registro transmisor



CS	RS	Operación	Registro seleccionado
0	x	x	Ninguno: canal de datos en alta impedancia
1	0	WR	Registro de transmisor
1	1	WR	Registro de control
1	0	RD	Registro receptor
1	1	RD	Registro de estado

Figura 11-8 Diagrama de bloque de una interface típica de comunicación asíncrona.

acepta un byte de datos de la CPU a través del canal de datos. Este byte se transfiere a un registro de corrimiento para transmisión serial. La parte receptora recibe información serial dentro de otro registro de corrimiento y, cuando se acumula un byte de datos completo, se transfiere al registro receptor. La CPU puede seleccionar que el registro receptor lea el byte a través del canal de datos. Los bits en el registro de estado se utilizan para banderas de entrada y salida y para registrar ciertos errores que pueden ocurrir durante la transmisión. La CPU puede leer el registro de estado para comprobar el estado de los bits de bandera y determinar si ha ocurrido algún error. Las líneas de control de selección de integrado y de lectura y escritura

sión son el error de paridad, el error de configuración y el error de superposición. Un error de paridad ocurre si la cantidad de dígitos 1 en los datos recibidos no corresponde a la paridad correcta. Un error de posición ocurre si no se detecta el número correcto de bits de alto al final del carácter recibido. Un error de superposición ocurre si la CPU no lee el carácter del registro receptor antes de que el siguiente quede disponible en el registro de corrimiento. Los errores de estancamiento dan como resultado una pérdida de caracteres en el flujo de datos recibido.

Buffer primero en entrar, primero en salir

FIFO

Un buffer primero en entrar, primero en salir (*first-in, first-out*, FIFO) es una unidad de memoria de localidades adyacentes que almacena información de manera que el primer dato que entra es el primero que sale. Un búffer FIFO tiene terminales de entrada y salida separadas. La característica importante de este buffer es que puede introducir y sacar datos a dos velocidades diferentes y que los datos de salida están siempre en el mismo orden en el cual se introdujeron al buffer. Cuando se coloca entre dos unidades, el FIFO puede aceptar datos de la unidad fuente a una velocidad de transferencia y enviar los datos a la unidad destino a otra velocidad. Si la unidad fuente es más lenta que la unidad destino, el búffer puede llenarse con datos a una velocidad lenta y después vaciarse a una velocidad más rápida. Si la fuente es más rápida que el destino, el FIFO es útil para aquellos casos en donde los datos fuente arriban en grandes cantidades que llenan el buffer, pero que el tiempo entre esos arribos es suficientemente largo para que la unidad destino vacie alguna o toda la información del buffer. Por lo tanto, un buffer FIFO puede ser útil en algunas aplicaciones cuando se transfieren datos en forma asíncrona. El buffer FIFO apila los datos conforme llegan y los entrega en el mismo orden cuando se necesitan.

El diagrama lógico de un buffer FIFO típico 4×4 se muestra en la figura 11-9. Consta de cuatro registros de 4 bits R_I , $I = 1, 2, 3, 4$ y un registro de control con flip-flops F_i , $i = 1, 2, 3, 4$, uno para cada registro. El FIFO puede almacenar cuatro palabras de cuatro bits cada una. Puede aumentarse la cantidad de bits por palabra al elevar la cantidad de bits en cada registro y puede aumentarse la cantidad de palabras al elevar la cantidad de registros.

Un flip-flop F_i en el registro de control que está activo en 1, indica que una palabra de datos de 4 bits está almacenada en el registro R_I correspondiente. Un 0 en F_i indica que el registro correspondiente no contiene datos válidos. El registro de control dirige el movimiento de datos por los registros. Cada vez que el bit F_i del registro de control está activado ($F_i = 1$) y el bit F_{i+1} se reactiva ($F'_{i+1} = 1$), se genera un pulso de reloj que hace que el registro $R_{(I+1)}$ acepte los datos del registro R_I . La misma transición de reloj activa F_{i+1} en 1 y desactiva F_i en 0. Esto provoca que la bandera de control se mueva una posición a la derecha junto con los datos. Los datos en el registro se mueven hacia abajo del FIFO, hacia la salida, mientras existan posiciones

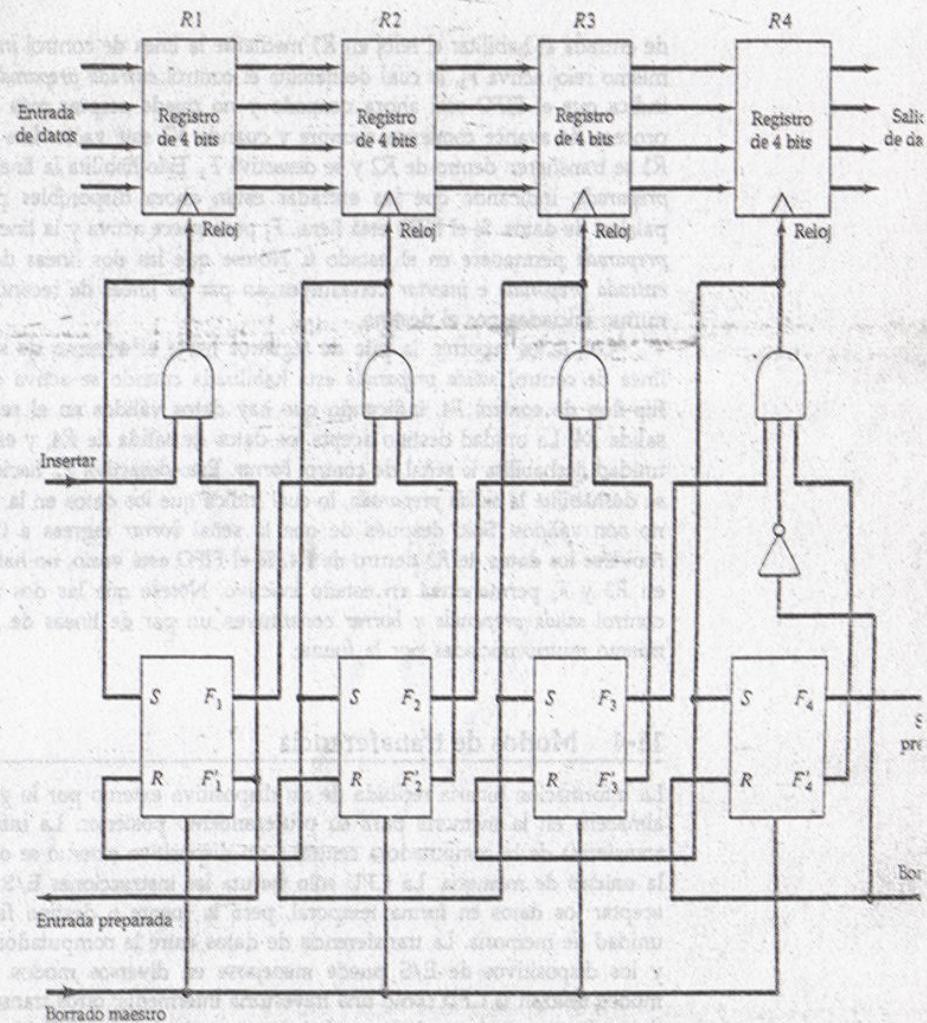


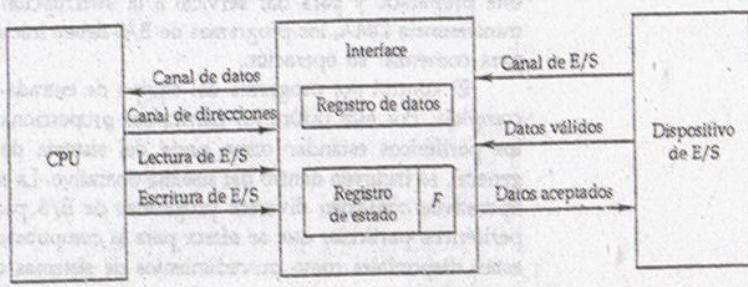
Figura 11-9 Diagrama de circuito de buffer FIFO 4 × 4.

vacías adelante de él. Esta operación de avance se detiene cuando los datos alcanzan el registro R_1 con el siguiente flip-flop F_{n1} activado en 1 o cuando alcanzan el último registro R_4 . Se utiliza un borrado general maestro para inicializar todos los flip-flops del registro de control en 0.

Se insertan datos dentro del buffer siempre y cuando esté habilitada la señal *entrada preparada*. Esto ocurre cuando se desactiva el primer flip-flop F_1 , indicando que el registro R_1 está vacío. Se cargan los datos de las líneas

requiere que la CPU ejecute varias instrucciones, incluyendo una instrucción de entrada para transferir los datos del dispositivo a la CPU y una instrucción de almacenamiento para transferir los datos de la CPU a la memoria. Pueden necesitarse otras instrucciones para verificar que están disponibles los datos del dispositivo y para contar la cantidad de palabras transferidas.

Un ejemplo de transferencia de datos de un dispositivo de E/S por medio de una interface a la CPU se muestra en la figura 11-10. El dispositivo transfiere bytes de datos uno a la vez, conforme están disponibles. Cuando está disponible un byte de datos, el dispositivo lo coloca en el canal de E/S y habilita su línea de datos válidos. La interface acepta el byte en su registro de datos y habilita la línea de datos aceptados. La interface activa un bit en el registro de estado que denominaremos bit de "bandera" o bit F. Ahora el dispositivo puede deshabilitar la línea de datos válidos, pero no transferirá otro byte hasta que la interface deshabilite la línea de datos aceptados. Esto se apegue al procedimiento de reconocimiento mutuo establecido en la figura 11-5.



F = Bit de bandera

Figura 11-10 Transferencia de datos de dispositivo E/S a CPU.

Está escrito un programa para la computadora con el fin de comprobar la bandera en el registro de estado, para determinar si se ha colocado un byte en el registro de datos mediante el dispositivo de E/S. Esto se hace al leer el registro de estado dentro del registro de la CPU y comprobar el valor del bit de bandera. Si la bandera es igual a 1, la CPU lee los datos del registro de datos. Después, la CPU o la interface desactivan el bit de bandera a 0, dependiendo de cómo están diseñados los circuitos de la interface. Una vez que se desactiva la bandera, la interface deshabilita la línea de datos aceptados y el dispositivo puede transferir, entonces, el siguiente byte de datos.

Un diagrama de flujo del programa que debe escribirse para la CPU, se muestra en la figura 11-11. Se considera que el dispositivo está enviando

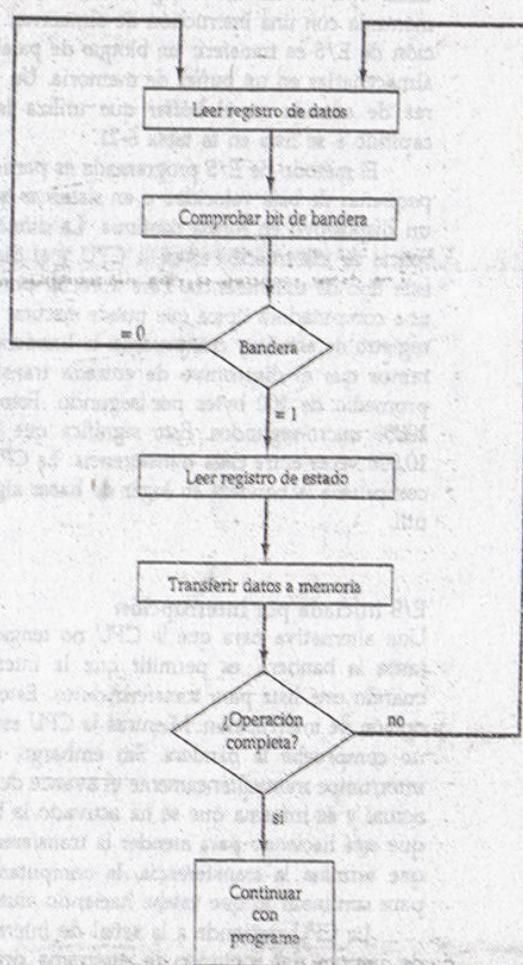


Figura 11-11 Diagrama de flujo para programa de CPU para introducir datos.

una secuencia de bytes que debe almacenarse en la memoria. La transferencia de cada byte requiere de tres instrucciones:

1. Leer el registro de estado.
2. Comprobar el estado del bit de bandera y transferir el control al paso 1 si no está activado o al paso 3 si lo está.
3. Leer el registro de datos.

Cada byte se lee en el registro de la CPU y después se transfiere a la memoria con una instrucción de almacenar. Una tarea común de programación de E/S es transferir un bloque de palabras de un dispositivo de E/S y almacenarlas en un buffer de memoria. Un programa que almacena caracteres de entrada en el búffer que utiliza las instrucciones definidas en el capítulo 6 se lista en la tabla 6-21.

El método de E/S programada es particularmente útil en computadoras pequeñas de baja velocidad o en sistemas que están dedicados a monitorear un dispositivo en forma continua. La diferencia en la velocidad de transferencia de información entre la CPU y el dispositivo de E/S hace inefficiente este tipo de transferencia. Para apreciar por qué es inefficiente, consideremos una computadora típica que puede ejecutar las dos instrucciones que leen el registro de estado y comprueban la bandera en un micro-segundo. Consideremos que el dispositivo de entrada transfiere sus datos a una velocidad promedio de 100 bytes por segundo. Esto es equivalente a un byte cada 10000 micro-segundos. Esto significa que la CPU comprobará la bandera 10,000 veces entre cada transferencia. La CPU está gastando tiempo mientras comprueba la bandera en lugar de hacer alguna otra tarea de procesamiento útil.

E/S iniciada por interrupción

Una alternativa para que la CPU no tenga que monitorear en forma constante la bandera, es permitir que la interfaz informe a la computadora cuando esté lista para transferir datos. Este modo de transferencia utiliza la opción de interrupción. Mientras la CPU está corriendo un programa común no comprueba la bandera. Sin embargo, cuando se activa la bandera, se interrumpe momentáneamente el avance de la computadora con el programa actual y se informa que se ha activado la bandera. La CPU se desvía de lo que está haciendo para atender la transferencia de entrada o salida. Después que termina la transferencia, la computadora retorna al programa previo para continuar lo que estaba haciendo antes de la interrupción.

La CPU responde a la señal de interrupción al almacenar la dirección de retorno del contador de programa dentro de una pila de memoria y después transferirse el control a una rutina de servicio que procesa la transferencia de E/S requerida. La manera en que el procesador elige la dirección de transferencia a la rutina de servicio varía de una unidad a otra. En principio, existen dos métodos para lograr esto. Uno se llama *interrupción con vector* y el otro *interrupción sin vector*. En una interrupción sin vector, la dirección de transferencia de control del programa se asigna a una posición física en la memoria. En una interrupción con vector, la fuente que interrumpe proporciona la información de la transferencia a la computadora. Esta información se llama el *vector de interrupción*. En algunas computadoras el vector de interrupción es la primera dirección de la rutina de servicio de E/S. En otras computadoras el vector de interrupción es una dirección que

nueva PSW para la rutina de servicio. Aquí no consideramos la PSW para no complicar el análisis de las interrupciones de E/S.

En una aplicación típica, se conectan varios dispositivos de E/S a la computadora, y cada dispositivo puede originar una solicitud de interrupción. La primera tarea del sistema de interrupción es identificar la fuente de la interrupción. También existe la posibilidad de que varias fuentes soliciten servicio en forma simultánea. En este caso, el sistema debe decidir también a cuál dispositivo atender primero.

prioridad de interrupción Una prioridad de interrupción es un sistema que establece una prioridad entre las diversas fuentes para determinar qué condición se va atender primero cuando llegan al mismo tiempo dos solicitudes. El sistema también puede determinar cuales condiciones se permiten para interrumpir a la computadora mientras se da servicio a otra interrupción. Se asignan niveles de interrupción de alta prioridad a solicitudes que, si se posponen o interrumpen, pueden producir consecuencias serias. Los dispositivos con transferencias de alta velocidad como discos magnéticos reciben una alta prioridad y los dispositivos lentos como los teclados reciben baja prioridad. Cuando dos dispositivos interrumpen la computadora al mismo tiempo, la computadora atiende al dispositivo con mayor prioridad.

encuesta Puede establecerse la prioridad de interrupciones simultáneas mediante programación o circuitería. Se usa un procedimiento de "encuesta" para identificar la fuente de prioridad más alta por medio de programación. En este método existe una dirección de transferencia de control común para todas las interrupciones y el programa cuida que las interrupciones comiencen en la dirección de transferencia y registra las fuentes de interrupción en secuencia. El orden en la cual se prueba determina la prioridad de cada interrupción. Se prueba primero la fuente de prioridad más alta y, si su señal de interrupción está activada, el control se transfiere a una rutina de servicio para esta fuente. De otra manera, se prueba la fuente con la siguiente prioridad hacia abajo y así sucesivamente. Por lo tanto, la rutina de servicio inicial para todas las interrupciones consiste en un programa que prueba las fuentes de interrupción en secuencia y transfiere el control a una de varias rutinas de servicio posibles. La rutina de servicio particular alcanzada, pertenece al dispositivo de prioridad más alta entre todos los dispositivos que interrumpieron a la computadora. La desventaja del método de programación es que, si hay muchas interrupciones, el tiempo requerido para registrarlas puede exceder el tiempo disponible para atender el dispositivo de E/S. En esta situación, puede utilizarse una unidad de circuito de prioridad de interrupción para acelerar la operación.

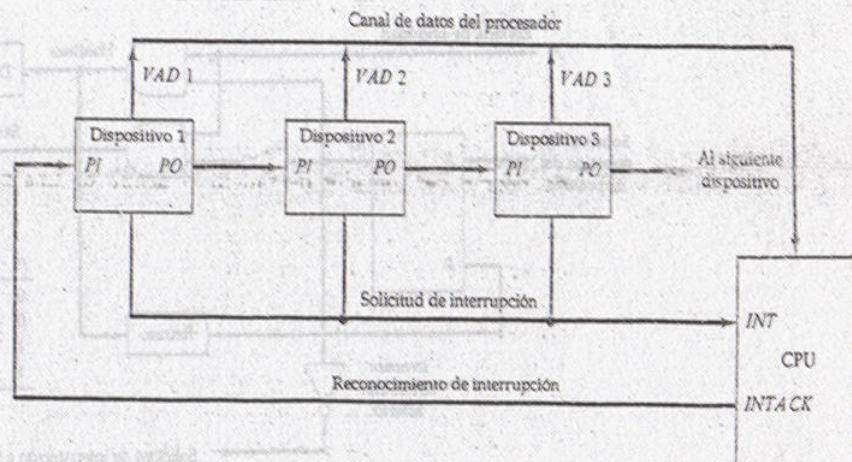
Una unidad de prioridad de interrupción de circuitería funciona como un administrador general en un ambiente de sistema de interrupciones. Acepta solicitudes de interrupción de muchas fuentes, determina cuál de las solicitudes que llegan tiene la prioridad más alta y emite una solicitud de interrupción a la computadora con base en esta determinación. Para acelerar la operación, cada fuente de interrupción tiene su propio vector de interrupción.

ción para accesar en forma directa su propia rutina de servicio. Por lo tanto, no se necesita el registro porque todas las decisiones las establece la unidad de circuito de prioridad de interrupción. La función de prioridad de circuitaria puede establecerla una conexión serial o paralela de líneas de interrupción. La conexión serial también se conoce como el método de cadena circular.

Prioridad de cadena de margaritas

El método de cadena de margaritas (daisy chain) de establecer prioridad consiste en una conexión serial de todos los dispositivos que solicitan una interrupción. El dispositivo con la prioridad más alta se coloca en la primera posición, seguido por los dispositivos de prioridad inferior hasta el dispositivo con la más baja prioridad, que se coloca al último en la cadena. Este método de conexión entre tres dispositivos en la CPU se muestra en la figura 11-12. La línea de solicitud de interrupción es común a todos los dispositivos y forma una conexión de lógica alambrada. Si cualquier dispositivo tiene su señal de interrupción en el estado de nivel bajo, la línea de interrupción va al estado de nivel bajo y habilita la entrada de interrupción en la CPU. Cuando no está pendiente ninguna interrupción, la línea de interrupción permanece en el estado de nivel alto y la CPU no reconoce interrupciones. Esto es equivalente a una operación OR lógica negativa. La CPU responde a una solicitud de interrupción al habilitar la línea de reconocimiento de interrupción. El dispositivo 1 recibe esta señal en su entrada de prioridad (*PI*). La señal de reconocimiento pasa al siguiente dispositivo por medio de

Figura 11-12 Interrupción con prioridad por cadena de margaritas.



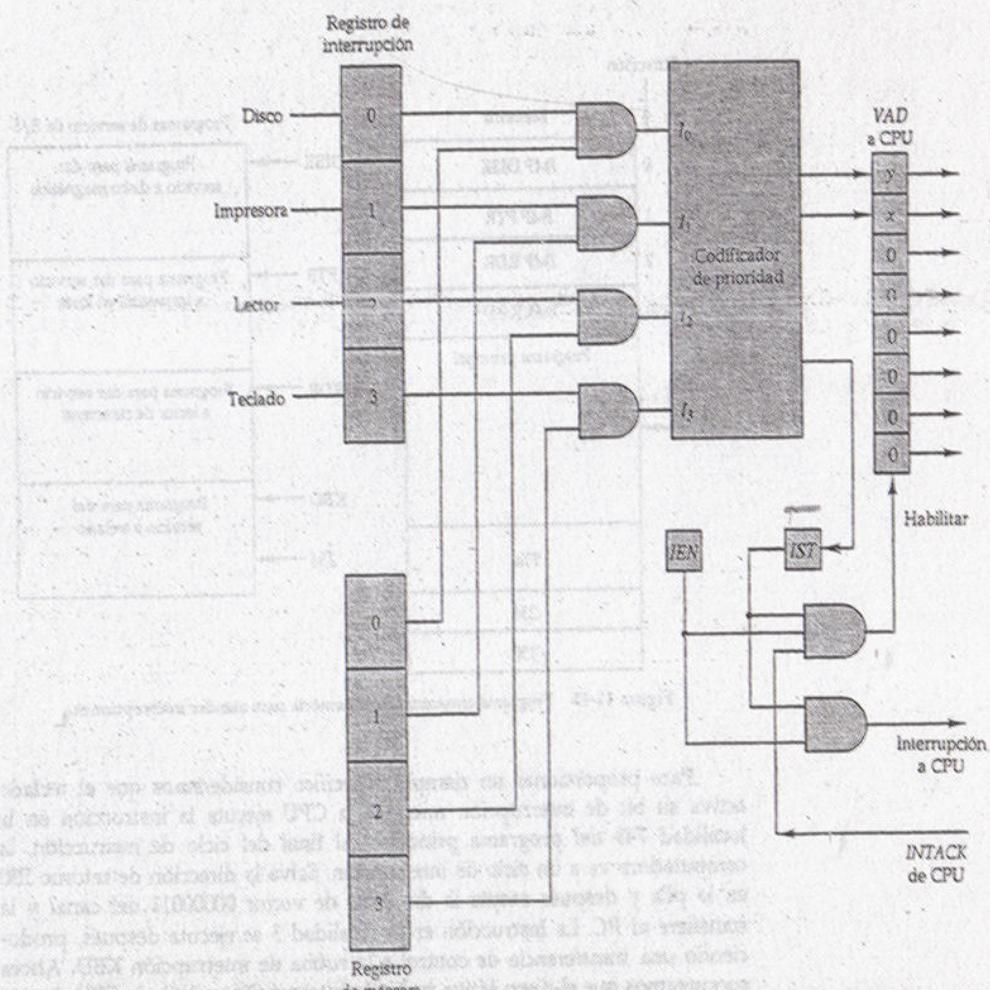


Figura 11-14 Circuitería de prioridad de interrupción.

ahora explicaremos el circuito codificador de prioridad y después analizaremos la interacción entre el controlador de prioridad de interrupción y la CPU.

Codificador de prioridad

El codificador de prioridad es un circuito que implementa la función de prioridad y desactivación. La lógica del codificador de prioridad es que, si dos o más entradas

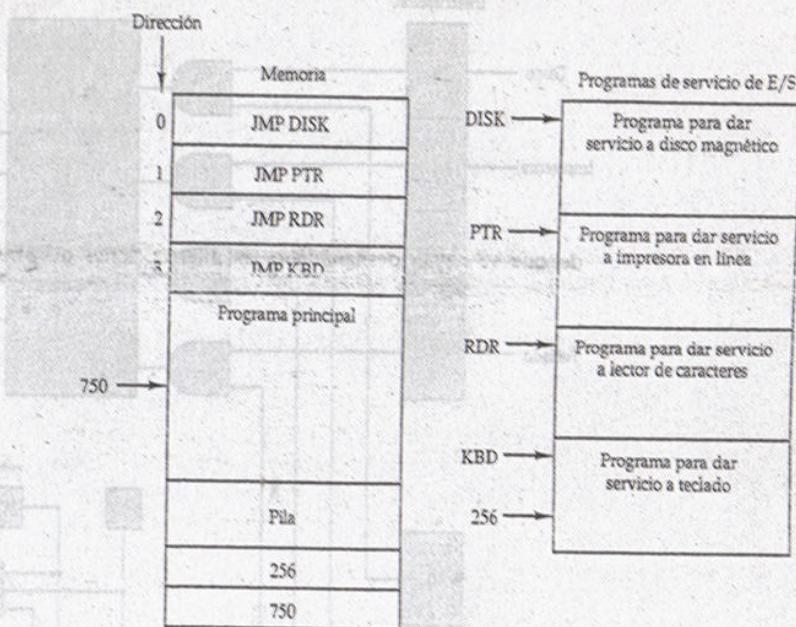


Figura 11-15 Programa almacenado en memoria para atender interrupciones.

Para proporcionar un ejemplo específico consideremos que el teclado activa su bit de interrupción mientras la CPU ejecuta la instrucción en la localidad 749 del programa principal. Al final del ciclo de instrucción, la computadora va a un ciclo de interrupción. Salva la dirección de retorno 750 en la pila y después acepta la dirección de vector 00000011 del canal y la transfiere al PC. La instrucción en la localidad 3 se ejecuta después, produciendo una transferencia de control a la rutina de interrupción KBD. Ahora supongamos que el disco activa su bit de interrupción cuando la CPU ejecuta la instrucción en la dirección 255 en el programa KBD. La dirección 256 se salva dentro de la pila y el control se transfiere al programa de servicio DISK. La última instrucción en cada rutina es un retorno de la instrucción de interrupción. Cuando termina el programa de servicio del disco, la instrucción de retorno se recupera de la pila y coloca 256 en el PC. Esto retorna el control a la rutina KBD para que continúe atendiendo el teclado. Al final del programa KBD, la última instrucción recupera la dirección de retorno de la pila y retorna el control al programa principal en la dirección 750. Por lo tanto, un dispositivo de prioridad mayor puede interrumpir a un dispositivo de prioridad menor. Se considera que el tiempo para dar servicio a una interrupción de alta prioridad es corto en comparación con la velocidad de

transferencia del dispositivo de baja prioridad, por lo que no ocurre ninguna pérdida de información.

Operaciones inicial y final

Cada rutina de servicio de interrupción debe tener un conjunto de operaciones inicial y final para controlar los registros en el sistema de circuitería de interrupción. Recuerde que la habilitación de interrupción *IEN* se desactiva al final de un ciclo de interrupción. Este flip-flop se debe activar de nuevo para habilitar solicitudes de interrupción de prioridad mayor, pero no antes de que se hayan deshabilitado las interrupciones de prioridad menor. La secuencia inicial de cada rutina de servicio de interrupción debe contener instrucciones para controlar la circuitería de interrupción de la siguiente manera:

1. Desactivar los bits del registro de máscara de prioridad inferior.
2. Desactivar el bit de estado de interrupción *IST*.
3. Salvar el contenido de los registros del procesador.
4. Activar el bit de habilitación de interrupción *IEN*.
5. Proceder con la rutina de servicio.

Los bits del registro de máscara de nivel inferior (incluye el bit de la fuente que interrumpió) se desactivan para evitar que estas condiciones habiliten la interrupción. Aunque a las fuentes de interrupción de prioridad menor se les asignan bits de número mayor en el registro de máscara, la prioridad puede cambiarse si se desea, porque el programa puede utilizar cualquier configuración de bits para el registro de máscara. El bit de estado de interrupción debe desactivarse para que pueda volverse a activar cuando ocurra una interrupción de prioridad mayor. El contenido de los registros de procesador se salvan porque puede necesitarlos el programa que se ha interrumpido después de que el control retorne a él. Después, se activa la habilitación de interrupción *IEN* para permitir que otras interrupciones (de prioridad mayor) y la computadora procedan a atender la solicitud de interrupción.

La secuencia final de cada rutina de servicio de interrupción debe contener instrucciones para controlar la circuitería de interrupción de la siguiente manera:

1. Activar el bit de habilitación de interrupción *IEN*.
2. Recuperar el contenido de los registros del procesador.
3. Desactivar el bit en el registro de interrupción que pertenece a la fuente que ha sido atendida.
4. Reactivar los bits de prioridad de nivel menor en el registro de cubierta.
5. Restablecer la dirección de retorno dentro del PC y establecer *IEN*.

como registros de interface de E/S. Por lo tanto, la CPU puede leer o escribir dentro de los registros DMA bajo el control de programa mediante el canal de datos.

Primero, la CPU inicializa el DMA. Después de eso, el DMA empieza y continúa la transferencia de datos entre la memoria y la unidad periférica hasta que se transfiere un bloque completo. El proceso de inicialización es esencialmente un programa que consiste en instrucciones de E/S que incluyen la dirección para seleccionar registros DMA particulares. La CPU inicializa el DMA al enviar la siguiente información por el canal de datos:

1. La dirección inicial del bloque de memoria en donde están disponibles los datos (para lectura) o donde se van a almacenar los datos (para escritura).
2. La cuenta de palabras, que es el número de palabras en el bloque de memoria.
3. Un control para especificar el modo de transferencia como de lectura o de escritura.
4. Un control para iniciar la transferencia DMA.

La dirección inicial se almacena en el registro de direccionamiento. La cuenta de palabras se almacena en el registro de cuenta de palabras y la información de control en el registro de control. Una vez que se inicializa el DMA, la CPU detiene la comunicación con el DMA, sólo que reciba una señal de interrupción o que desee comprobar cuántas palabras se han transferido.

Transferencia DMA

La posición del controlador DMA entre los otros componentes en un sistema de computadora se ilustra en la figura 11-18. La CPU se comunica con el DMA mediante los canales de dirección y de datos como con cualquier unidad de interface. El DMA tiene su propia dirección, la cual activa las líneas DS y RS. La CPU inicializa el DMA por medio del canal de datos. Una vez que el DMA recibe el comando de control de inicio, puede comenzar la transferencia entre el dispositivo periférico y la memoria.

Cuando el dispositivo periférico envía una solicitud al DMA, el controlador de DMA activa la línea BR, informando a la CPU que ceda los canales. La CPU responde con su línea BG, informando al DMA que sus canales están deshabilitados. Después el DMA pone el valor de su registro de direccionamiento dentro del canal de dirección, activa la señal RD o WR y envía un reconocimiento DMA al dispositivo periférico. Nótese que las líneas RD y WR en el controlador DMA son bidireccionales. La dirección de la transferencia depende del estado de la línea BG. Cuando BG = 0, RD y WR son líneas de entrada que permiten a la CPU comunicarse con los registros internos del DMA. Cuando BG = 1, RD y WR son líneas de salida del

cuenta de palabras no alcanza cero, el DMA comprueba la línea de solicitud que proviene del periférico. Para un dispositivo de alta velocidad, la línea estará activa tan pronto como se termine la transferencia previa. Después se inicia una segunda transferencia, y el proceso continúa hasta que se ha transferido todo el bloque. Si la velocidad del periférico es más lenta, la línea de solicitud DMA puede venir un poco más tarde. En este caso, el DMA deshabilita la línea de solicitud de canal para que la CPU pueda continuar ejecutando su programa. Cuando el periférico solicita una transferencia, el DMA vuelve a solicitar los canales.

Si el registro de cuenta de palabras llega a cero, el DMA detiene cualquier transferencia posterior y retira su solicitud de canal. También informa a la CPU de la terminación mediante una interrupción. Cuando la CPU responde a la interrupción, lee el contenido del registro de cuenta de palabras. El valor cero de este registro indica que todas las palabras se transfirieron exitosamente. La CPU puede leer este registro en cualquier momento para comprobar la cantidad de palabras que ya se han transferido.

Un controlador DMA puede tener más de un canal. En este caso, cada canal tiene un par de señales de control para señales de solicitud y reconocimiento que están conectadas a dispositivos periféricos separados. Cada canal también puede tener su propio registro de direccionamiento y registro de cuenta de palabras dentro del controlador DMA. Puede establecerse una prioridad entre los canales para que los canales con alta prioridad sean atendidos antes que los canales de prioridad menor.

La transferencia DMA es muy útil en muchas aplicaciones. Se utiliza para transferencias rápidas de información entre discos magnéticos y memoria. También es útil para actualizar la pantalla en una terminal interactiva. De manera típica, una imagen de la pantalla de la terminal se conserva en la memoria y puede actualizarse bajo el control del programa. El contenido de la memoria puede transferirse a la pantalla en forma periódica, mediante una transferencia de DMA.

11-7 Procesador de entrada-salida (IOP)

En lugar que cada interface se comunique con la CPU, una computadora puede incorporar uno o más procesadores externos y asignarles la tarea de comunicarse directamente con todos los dispositivos E/S. Un procesador de entrada-salida (IOP), puede clasificarse como un procesador con capacidad de acceso directo a memoria que comunica con dispositivos de E/S. En esta configuración, el sistema de computadora puede dividirse en una unidad de memoria y varios procesadores que comprenden la CPU y uno o más IOP. Cada IOP atiende tareas de entrada y salida, relevando a la CPU de los "quehaceres domésticos" que involucran las transferencias de E/S. Un procesador que comunica con terminales remotas por teléfono y otros medios de comunicación en forma serial se llama un procesador de comunicación de datos (DCP).

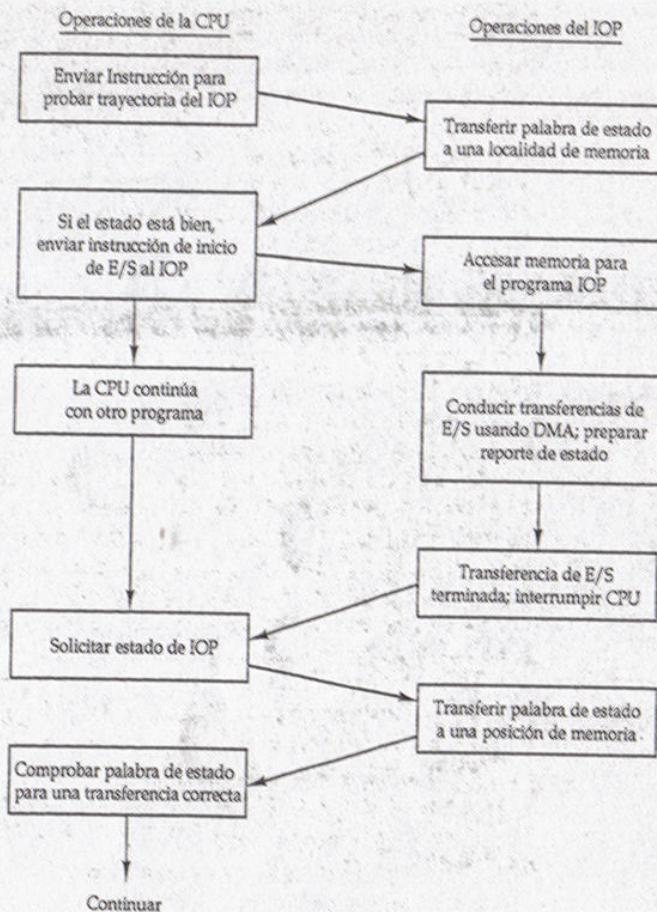


Figura 11-20 Comunicación CPU-IOP.

la transferencia de E/S. La dirección de memoria recibida con esta instrucción le dice al IOP en dónde encontrar su programa.

Ahora la CPU puede continuar con otro programa mientras el IOP está ocupado con el programa de E/S. Ambos programas hacen referencia a la memoria mediante una transferencia DMA. Cuando el IOP termina la ejecución de su programa, envía una solicitud de interrupción a la CPU. La CPU responde a la interrupción al emitir una instrucción para leer el estado del IOP. El IOP responde al colocar el contenido de su reporte de estado dentro de una localidad de memoria específica. La palabra de estado indica si ha terminado la transferencia o si ha ocurrido cualquier error durante ella. De

condición es diferente para cada instrucción de E/S. Pero, en general, especifican si el canal o el dispositivo están ocupados, si son operacionales o no, si hay interrupciones pendientes, si la operación de E/S comenzó exitosamente y si se almacenó una palabra de estado en la memoria por medio del canal.

El formato de la palabra de estado del canal se muestra en la figura 11-21(b). Siempre está almacenado en la posición 64 en la memoria. El campo de clave es un mecanismo de protección utilizado para evitar el acceso no autorizado por parte de un usuario a la información que pertenece a otro usuario o al sistema operativo. El campo de dirección de la palabra de estado proporciona la dirección de la última palabra del comando utilizada por el canal. El campo de cuenta proporciona la cuenta residual cuando se terminó la transferencia. El campo de cuenta mostrará cero si la transferencia se terminó exitosamente. El campo de estado identifica las condiciones en el dispositivo y el canal y cualesquiera errores ocurridos durante la transferencia.

La diferencia entre las instrucciones iniciar E/S e iniciar envío rápido de E/S es que la última requiere menos tiempo de CPU para su ejecución, cuando el canal recibe una de estas dos instrucciones, hace referencia a la localidad 72 de la memoria para la dirección de la primera palabra de comando del canal (CCW). El formato de la palabra de comando del canal se muestra en la figura 11-21(c). El campo de dirección de datos especifica la primera dirección de un búfer de memoria y el campo de cuenta proporciona la cantidad de bits que participan en la transferencia. El campo de comando especifica una operación de E/S y los bits de bandera proporcionan

Figura 11-21 Relación de formatos de palabra: E/S de la IBM 370.

Código de operación	Dirección de canal	Dirección de dispositivo
---------------------	--------------------	--------------------------

a) Formato de instrucciones de E/S

Clave	Dirección	Estado	Cuenta
-------	-----------	--------	--------

b) Formato de palabra de estado del canal

Código de comando	Dirección de datos	Banderas	Cuenta
-------------------	--------------------	----------	--------

c) Formato de palabra de comando del canal

módem receptor de las transiciones de señal que ocurren en los datos recibidos. Cualquier cambio de secuencia que puede ocurrir entre los relojes transmisor y receptor se ajusta continuamente al mantener el reloj receptor en la frecuencia del flujo de bits que llega. El modem transfiere los datos recibidos junto con el reloj a la unidad de interface. La interface o terminal en el lado transmisor también utiliza la información de reloj de su módem. De esta manera, se mantiene la misma velocidad de bits en el transmisor y en el receptor.

A diferencia de la transmisión asíncrona, en la cual cada carácter puede enviarse en forma separada con sus propios bits de inicio y paro, la transmisión síncrona debe enviar un mensaje continuo para mantener la sincronización. El mensaje consta de un grupo de bits transmitidos en forma secuencial como un bloque de datos. Todo el bloque se transmite con caracteres de control especiales al comienzo y al final del bloque. Los caracteres de control al inicio del bloque proporcionan la información necesaria para separar los bits que llegan en caracteres individuales.

Una de las funciones del procesador de comunicación de datos es verificar la presencia de errores de transmisión. Puede detectarse un error al comprobar la paridad en cada carácter recibido. Otro procedimiento utilizado en terminales asíncronas en las que interviene un operador humano es escuchar el eco de los caracteres. El procesador reconoce los caracteres transmitidos del teclado a la computadora y los retransmite a la impresora terminal. El operador debe darse cuenta de que ha ocurrido un error durante la transmisión si el carácter impreso no es igual a la tecla que se ha oprimido.

transferencia de bloque

En la transmisión síncrona, en la cual se transmite un bloque de caracteres completo, cada carácter tiene un bit de paridad que debe verificar el receptor. Después de que se envía el bloque completo, el transmisor envía un carácter más que constituye una paridad sobre la longitud del mensaje. Este carácter se llama una comprobación de redundancia longitudinal (*longitudinal redundancy check, LRC*) y es la acumulación de las OR exclusivas de todos los caracteres recibidos. La estación receptora calcula la LRC conforme recibe caracteres y la compara con la LRC transmitida. Las LRC calculadas deben ser iguales para que los mensajes no contengan error. Si el receptor encuentra un error en el bloque transmitido, le informa a quien envía que retransmita el mismo bloque de nuevo. Otro método utilizado para comprobar errores en la transmisión es la comprobación de redundancia cíclica (*cyclic redundancy check, CRC*). Este es un código de polinomio que se obtiene de los bits de mensaje al pasarlo a través de un registro de corrimiento retroalimentado que contiene varias compuertas OR exclusivas. Este tipo de código es conveniente para detectar errores de ráfaga que ocurren en el canal de comunicación.

CRC

Pueden transmitirse datos entre dos puntos en tres modos diferentes: simplex, semidúplex o dúplex completo. Una línea simplex transmite información sólo en una dirección. Este modo se utiliza rara vez en comunicación de datos porque el receptor no puede comunicarse con el transmisor para

TABLA 11-4 Caracteres ASCII de control de comunicación

Código	Símbolo	Significado	Función
0010110	SYN	Inactivo síncrono	Establece sincronismo
0000001	SOH	Inicio de encabezado	Encabezado de mensaje de bloque
0000010	STX	Inicio de texto	Antecede un bloque de texto
0000011	ETX	Fin de texto	Termina un bloque de texto
0001000	EOT	Fin de transmisión	Concluye una transmisión
0001100	ACK	Reconocimiento	Reconocimiento afirmativo
0010101	NAK	Reconocimiento negativo	Reconocimiento negativo
0000101	ENQ	Consulta	Consulta si la terminal está encendida
0010111	ETB	Fin de bloque de transmisión	Fin de bloque de datos
0010000	DLE	Escape de enlace de datos	Carácter de control especial

letras. El papel de carácter en el control de la transmisión de datos se explica en forma breve en la columna de función de la tabla.

El carácter SYN sirve como un agente de sincronización entre el transmisor y el receptor. Cuando se utiliza el código ASCII de 7 bits con un bit de paridad impar en la posición más significativa, el carácter SYN asignado tiene el código de 8 bits 00010110 que tiene la propiedad que, ante la ocurrencia de un corrimiento circular, se repite a sí mismo sólo después de un ciclo de 8 bits completo. Cuando el transmisor empieza a enviar caracteres de 8 bits, envía primero unos cuantos caracteres, y después envía el mensaje real. La serie continua de bits iniciales que acepta el receptor se comprueba en busca de un carácter SYN. En otras palabras, con cada pulso de reloj, el receptor comprueba los últimos 8 bits recibidos. Si no coinciden los bits del carácter SYN el receptor acepta el siguiente bit y rechaza el bit anterior de orden superior y vuelve a comprobar los últimos 8 bits recibidos en busca de un carácter SYN. Esto se repite después de cada pulso de reloj y después de cada bit recibido hasta que se reconoce un carácter SYN. Una vez que se detecta un carácter SYN, el receptor ha "armado" un carácter. De ahí en adelante el receptor cuenta cada 8 bits y los acepta como un sólo carácter. Por lo general, el receptor comprueba dos caracteres SYN consecutivos para que no exista la duda que el primero ocurrió como resultado de una señal de ruido en la línea. Además, cuando el transmisor está desocupado y no tiene ningunos caracteres de mensaje que enviar, emplea una serie continua de caracteres SYN. El receptor reconoce estos caracteres como una condición para sincronizar la línea y pasa a su estado inactivo síncrono. En este estado, las dos unidades son síncronas en bits y caracteres, aunque no se comunique información significativa.

Los mensajes se transmiten por el enlace de datos con un formato establecido que consiste en un campo de encabezado, un campo de texto y

EN ESTE CAPÍTULO

- 12-1 Jerarquía de la memoria
- 12-2 Memoria principal
- 12-3 Memoria auxiliar
- 12-4 Memoria asociativa
- 12-5 Memoria caché
- 12-6 Memoria virtual
- 12-7 Circuitería de administración de la memoria

12-1 Jerarquía de la memoria

La unidad de memoria es un componente esencial en cualquier computadora digital, porque se necesita para almacenar programas y datos. Una computadora muy pequeña con una aplicación limitada podría cumplir con la tarea que se pretende, sin necesidad de capacidad de almacenamiento adicional. La mayoría de las computadoras de propósito general correrían con mayor eficiencia si estuvieran equipadas con almacenamiento adicional, aparte de la capacidad de la memoria principal. Casi no hay suficiente espacio en una unidad de memoria para alojar todos los programas que se usan en una computadora típica. Además, la mayoría de los usuarios de computadora acumulan y siguen acumulando grandes cantidades de paquetes de programación de procesamiento de datos. Y el procesador no necesita toda la información almacenada al mismo tiempo. Por lo tanto, es más económico utilizar dispositivos de almacenamiento de bajo costo, que sirvan como un respaldo para almacenar la información que en ese momento no utiliza la CPU. La unidad de memoria que se comunica directamente con la CPU se llama *memoria principal*. Los dispositivos que proporcionan almacenamiento de respaldo se llaman *memoria auxiliar*. Los dispositivos de memoria auxiliar

memoria auxiliar

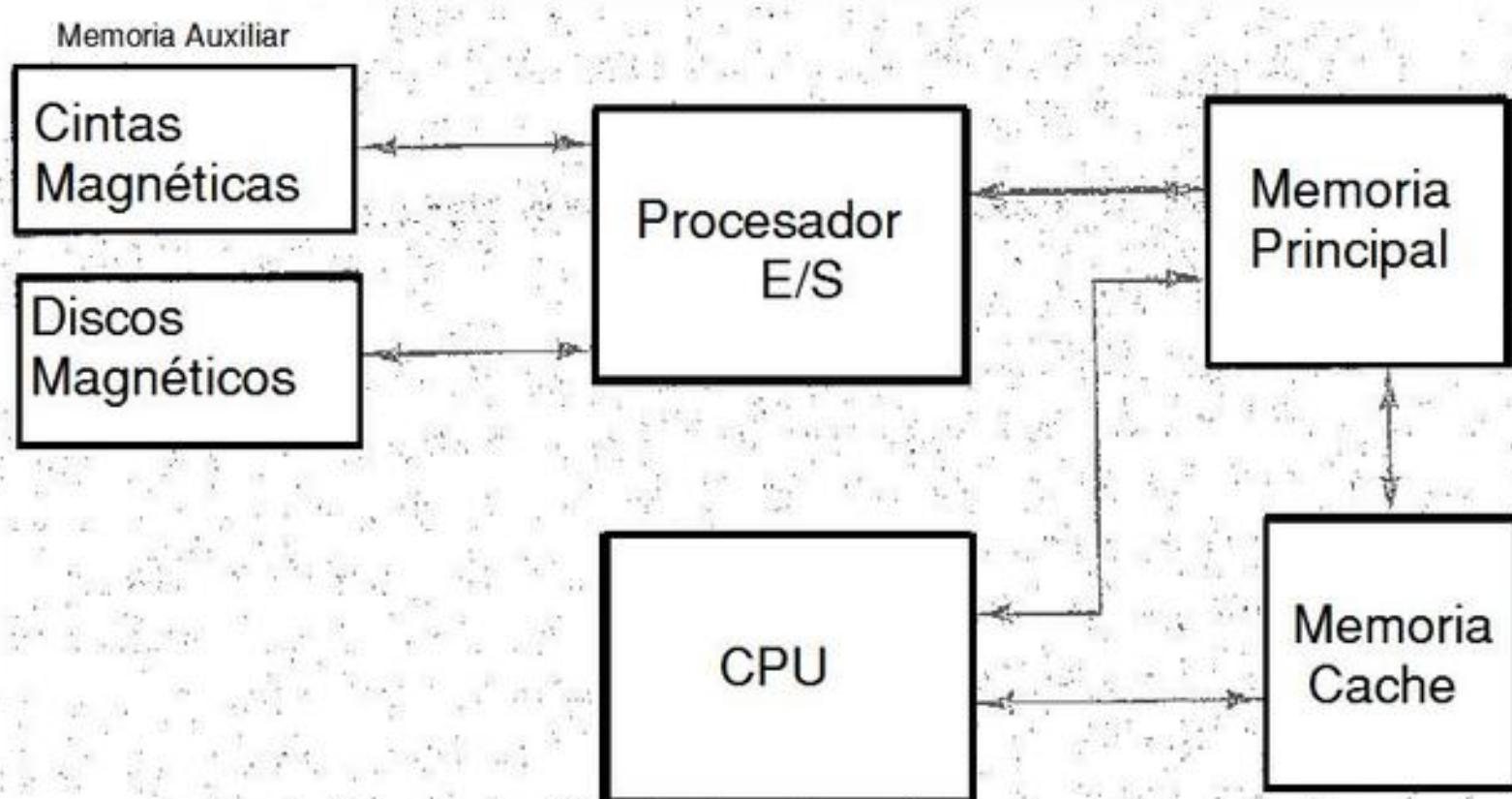
que se utilizan con más frecuencia en los sistemas de computadora son los discos y cintas magnéticas. Se usan para almacenar programas del sistema, grandes archivos de datos y otra información de respaldo. Sólo los programas y datos que necesita en ese momento el procesador residen en la memoria principal. Toda la demás información se almacena en la memoria auxiliar y se transfiere a la memoria principal cuando se necesita.

La capacidad total de memoria de una computadora puede considerarse como una jerarquía de componentes. El sistema de jerarquía de memoria consiste en todos los sistemas de almacenamiento que se emplean en un sistema de computadora; desde la memoria auxiliar, que es lenta pero de gran capacidad, hasta la memoria principal, que es relativamente más rápida o, tal vez, una memoria caché más rápida y pequeña, que resulta accesible para la lógica de procedimiento de alta velocidad. La figura 12-1 ilustra los componentes de una jerarquía típica de memoria. En la parte baja de la jerarquía están las cintas magnéticas, que son relativamente lentas y que se usan para almacenar archivos removibles. Despues, están los discos magnéticos que se utilizan como almacenamiento de respaldos. La memoria principal ocupa una posición central al poder comunicarse directamente con la CPU y con los dispositivos de memoria auxiliar, mediante un procesador de E/S. Cuando la CPU necesita los programas que no residen en la memoria principal, se transportan de la memoria auxiliar. Los programas que no necesita en ese momento la memoria principal, se transfieren dentro de la memoria auxiliar, para proporcionar espacio a los programas y datos que se usan en ese momento.

En ocasiones, se utiliza una memoria especial de muy alta velocidad, llamada *caché*, para aumentar la capacidad de procesamiento al poner disponible para la CPU los programas y datos actuales a una velocidad rápida.

memoria caché

Figura 12-1 Jerarquía de memoria en un sistema de computadora.



blecer la carga que se pierde. La RAM dinámica ofrece consumo de energía reducido y mayor capacidad de almacenamiento en un sólo C.I. de memoria. La RAM estática es más fácil de usar y tiene ciclos de lectura y escritura más cortos.

Gran parte de la memoria principal, en una computadora de propósito general, está formada de circuitos integrados de RAM, pero una parte de la memoria está formada por la ROM. Originalmente, se utilizaba RAM para hacer referencias a una memoria de acceso aleatorio, pero ahora se utiliza para representar una memoria de lectura/escritura y diferenciarla de una memoria de sólo lectura, aunque una ROM es también de acceso aleatorio. Se utiliza la RAM para almacenar la mayor parte de los procesos y datos que están sujetos a cambio. Se utiliza la ROM para almacenar programas que residen en forma permanente en la computadora y para tablas de constantes que no cambian de valor una vez que se termina la programación de la computadora.

Entre otras cosas, la parte ROM de la memoria se necesita para almacenar un programa inicial llamado *cargador de inicialización (bootstrap loader)*. El cargador de inicialización es un programa cuya función es iniciar la operación de la programación de la computadora cuando se enciende la unidad. Como la RAM es volátil, su contenido se destruye cuando se desconecta la corriente. El contenido de la ROM no se afecta después que se apaga y enciende la unidad. El arranque de una computadora consiste en encender la corriente y comenzar la ejecución de un programa inicial. Por lo tanto, cuando se enciende la corriente, la circuitería de la computadora establece el contador de programa en la primera dirección del cargador de inicialización. El programa de inicialización carga una parte del sistema operativo del disco a la memoria principal y después se transfiere el control al sistema operativo, el cual prepara la computadora para su uso general.

Los C.I. de RAM y ROM están disponibles en diferentes tamaños. Si la memoria que necesita la computadora es mayor que la capacidad que un C.I. es necesario combinar varios para formar el tamaño de memoria requerido. Para mostrar esta interconexión, pongamos un ejemplo de una memoria de 1024×8 construida con C.I. de RAM 128×8 y C.I. de ROM 512×8 .

C.I. de RAM y de ROM

Un C.I. RAM es más adecuado para la comunicación con la CPU si tiene una o más entradas de control que lo seleccionen sólo cuando se necesita. Otra característica común es un canal (o bus) de datos bidireccional que permite la transferencia de datos de la memoria a la CPU, durante una operación de lectura, o de la CPU a la memoria, durante una operación de escritura. Puede construirse un canal bidireccional con acopladores de tres estados. Puede colocarse la salida del acoplador de tres estados en uno de tres estados posibles. Una señal equivalente al 1 lógico, una señal equivalente al 0 lógico o un estado de alta impedancia. El 1 y el 0 lógicos son señales

memoria de sólo
lectura (ROM)

cargador de
inicialización

inicialización de
computadora

canal bidireccional

Una unidad de disco con discos removibles se llama de disco flexible. Los discos que se utilizan con una unidad de disco flexible son pequeños discos que se pueden quitar, hechos de plástico con una cubierta de material magnético de grabación. Por lo general se usan dos tamaños, con diámetros de 5.25 y 3.5 pulgadas. Los discos de 3.5 pulgadas son más pequeños y pueden almacenar más datos que los de 5.25 pulgadas. Los discos flexibles se usan ampliamente en las computadoras personales, como medio para distribuir programación a los usuarios.

Cinta magnética

Un transporte de cinta magnética consiste en componentes eléctricos, mecánicos y electrónicos para proporcionar las partes y el mecanismo de control para una unidad de cinta magnética. La cinta misma es una tira de plástico cubierta con un medio magnético de grabación. Los bits se graban sobre la cinta como puntos magnéticos, a lo largo de varias pistas. Por lo general, se graban en forma simultánea 7 o 9 bits para formar un carácter junto con un bit de paridad. En cada pista está montada una cabeza de lectura/escritura para que puedan grabarse y leerse los datos como una secuencia de caracteres.

Las unidades de cinta magnética pueden detenerse, prenderse para que avancen o se muevan en reversa, o pueden rebobinarse. Sin embargo, no pueden iniciarse o detenerse con la suficiente rapidez entre los caracteres individuales. Por esta razón, la información se graba en bloques que se denominan registros. Se insertan entre los registros porciones de cinta sin grabar, en las cuales se puede detener la cinta. La cinta comienza a moverse cuando está en una de esas porciones y mantiene su velocidad constante durante el tiempo que alcanza el siguiente registro. Cada registro en la cinta tiene un patrón de bits de identificación al comienzo y al final. Al leer el patrón de bits al principio, el control de cinta identifica el número de registro. Al leer el patrón de bits al final del registro, el control reconoce el comienzo de una porción sin grabar. Una unidad de cinta se dirige a especificar el número de registro y la cantidad de caracteres en él. Los registros pueden ser de longitud fija o variable.

12-4 Memoria asociativa

Muchas aplicaciones de procesamiento de datos requieren la búsqueda de conjuntos de datos en una tabla almacenada en la memoria. Un programa ensamblador busca la tabla de dirección de símbolos para extraer el equivalente binario del símbolo. Puede buscarse un número de cuenta en un archivo, para determinar el nombre del propietario y el estado de la cuenta. La manera establecida de buscar en una tabla es almacenar todos los datos donde puedan direccionarse en secuencia. El procedimiento de búsqueda es

*memoria
direccional
por contenido*

una estrategia para seleccionar una secuencia de direcciones, leer el contenido de la memoria en cada dirección y comparar la información que se lee con el dato que se busca, hasta que coincidan. La cantidad de accesos a memoria depende de la posición del conjunto de datos y de la eficiencia del algoritmo de búsqueda. Se han desarrollado muchos algoritmos de búsqueda para minimizar la cantidad de accesos mientras buscan un conjunto de datos en una memoria de acceso secuencial o aleatorio.

El tiempo requerido para encontrar un conjunto almacenado en la memoria puede reducirse mucho si pueden especificarse los datos almacenados, para su acceso, mediante el contenido de los datos mismos, en lugar de mediante una dirección. Una unidad de memoria accesada por su contenido se denomina *memoria asociativa* o *memoria direccionable por contenido* (Content Addressable Memory, CAM). Este tipo de memoria se accesa en forma simultánea y en paralelo, con base en el contenido de los datos, más que en la dirección o posición especificada. Cuando se escribe una palabra en una memoria asociativa no se proporciona una dirección. La memoria puede encontrar una posición vacía sin usar para almacenar la palabra. Cuando se va a leer una palabra de una memoria asociativa, se especifica el contenido de la palabra o parte de ella. La memoria localiza todas las palabras que coinciden con el contenido especificado y las marca para lectura.

Debido a esta organización, la memoria asociativa sólo es conveniente para hacer búsquedas paralelas mediante asociación de datos. Además, pueden hacerse búsquedas sobre una palabra completa o sobre un campo específico de una palabra. Una memoria asociativa es más cara que una memoria de acceso aleatorio porque cada celda debe tener capacidad de almacenamiento y circuitos lógicos para hacer coincidir su contenido con un argumento externo. Por esta razón, en las memorias asociativas se usan aplicaciones en las que el tiempo de búsqueda es muy importante y debe ser corto.

Organización de la circuitería

El diagrama de bloque de una memoria asociativa se muestra en la figura 12-6. Consta de un arreglo de memoria y la lógica para m palabras, con n bits por palabra. El registro de argumentos A y el registro de claves K tienen cada uno n bits, uno por cada bit de una palabra. El registro de coincidencia M tiene m bits, uno para cada palabra de memoria. Cada palabra en la memoria se compara en paralelo con el registro de argumentos. Las palabras que coinciden con los bits del registro de argumentos activan el bit correspondiente en el registro de coincidencia. Después del proceso de buscar coincidencias, los bits activados en el registro de coincidencia, indican que son iguales sus palabras correspondientes. Se hace la lectura mediante un acceso secuencial a memoria para aquellas palabras cuyos bits correspondientes en el registro de coincidencia se han activado.

El registro de claves proporciona una máscara para elegir un campo o clave particular en la palabra de argumento. Se compara todo el argumento

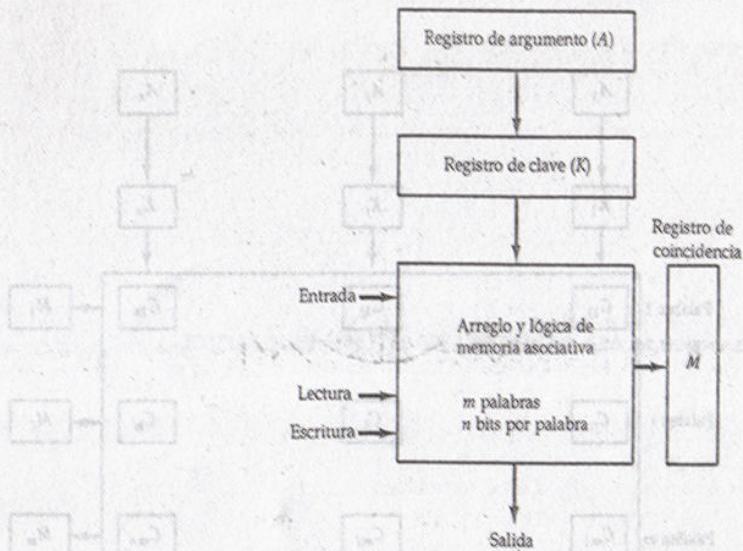


Figura 12-6 Diagrama de bloque de memoria asociativa.

con cada palabra de memoria si el registro de claves contiene sólo dígitos 1. De otra manera, se comparan sólo aquellos bits en el argumento que tienen dígitos 1 en la posición correspondiente del registro de claves. Por lo tanto, la clave proporciona una máscara o una pieza de información de identificación que especifica cómo se hace la referencia a memoria. Para mostrar un ejemplo numérico, supongamos que el registro de argumentos A y el registro de claves K tienen la configuración de bits que se muestra enseguida. Sólo se comparan los tres bits de la extrema izquierda de A con palabras de memoria, porque K tiene dígitos 1 en estas posiciones.

A	101 111100	
K	111 000000	
Palabra 1	100 111100	sin coincidencia
Palabra 2	101 000001	coincidencia

La palabra 2 coincide con el campo de argumento sin enmascarar porque son iguales los tres bits de la extrema izquierda del argumento y la palabra.

La relación entre el arreglo de memoria y los registros externos, en una memoria asociativa, se muestra en la figura 12-7. Se marcan las celdas en el arreglo mediante la letra C con dos subíndices. El primer subíndice proporciona el número de palabra y el segundo especifica la posición de bit en la palabra. Por lo tanto, la celda C_{ij} es la celda para el bit j en la palabra i . Un bit A_j en el registro de argumentos se compara con todos los bits en la

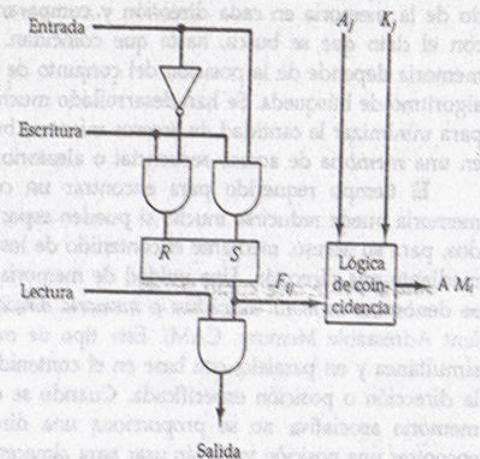


Figura 12-8 Una celda de memoria asociativa.

$$x_j = A_j F_{ij} + A'_j F'_{ij}$$

donde $x_j = 1$, si el par de bits en la posición j son iguales; de otra manera, $x_j = 0$.

Para que una palabra i sea igual al argumento en A debemos tener todas las variables x_j iguales a 1. Esta es la condición para activar el bit de coincidencia correspondiente M_i en 1. La función booleana para esta condición es:

$$M_i = x_1 x_2 x_3 \cdots x_n$$

y constituye la operación AND para todos los pares de bits que coinciden en una palabra.

Ahora incluimos el bit de clave K_j en la lógica de comparación. El requisito es que si $K_j = 0$, los bits correspondientes de A_j y F_{ij} no necesitan comparación. Sólo cuando $K_j = 1$, deben compararse. Este requisito se consigue al aplicar la función OR a cada término con K'_j , por lo tanto:

$$x_j + K'_j = \begin{cases} x_j & \text{if } K_j = 1 \\ 1 & \text{if } K_j = 0 \end{cases}$$

Cuando $K_j = 1$, tenemos K'_j y $x_j + 0 = x_j$. Cuando $K_j = 0$, entonces $K'_j = 1$ y $x_j + 1 = 1$. Un término $(x_j + K'_j)$ estará en el estado 1 si su par de bits no se comparan. Esto es necesario porque a cada término se le aplica una función AND con todos los otros términos para que una salida de 1 no tenga efecto. La comparación de los bits tiene efecto sólo cuando $K_j = 1$.

La lógica de coincidencia para la palabra i en la memoria asociativa puede expresarse ahora mediante la siguiente función booleana:

$$M_i = (x_1 + K'_1)(x_2 + K'_2)(x_3 + K'_3) \cdots (x_n + K'_n)$$

Cada término será igual a 1 si su correspondiente $K_j = 0$. Si $K_j = 1$, el término será 0 o 1, dependiendo del valor de x_j . Ocurrirá una coincidencia y M_i será igual a 1, si todos los términos son iguales a 1.

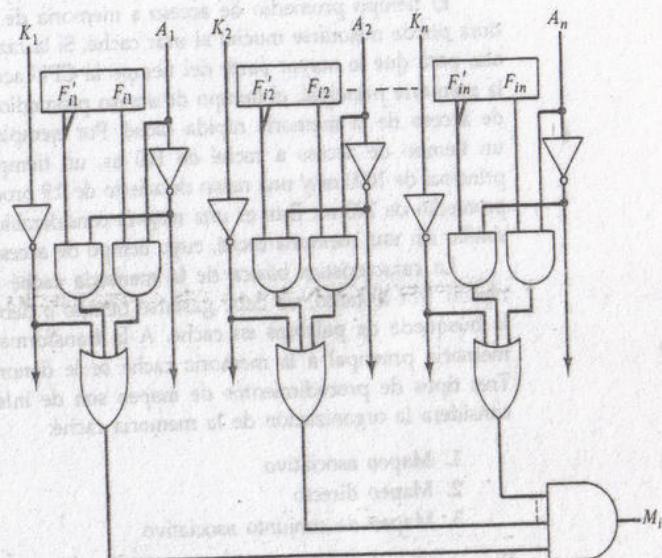
Si sustituimos la definición original de x_j , la función booleana anterior puede expresarse como sigue:

$$M_i = \prod_{j=1}^n (A_j F_{ij} + A'_j F'_{ij} + K'_j)$$

donde \prod es un símbolo de producto que representa la operación AND de todos los términos n . Necesitamos n de tales funciones, una para cada palabra $i = 1, 2, 3, \dots, m$.

El circuito para hacer coincidir una palabra se muestra en la figura 12-9. Cada celda requiere dos compuertas AND y una OR. Se necesitan inversores para A_j y K_j , uno para cada columna y se usan para todos los bits en la columna. La salida de todas las compuertas OR en las celdas de la misma

Figura 12-9 Lógica de coincidencia para una palabra de memoria asociativa.



palabra, van a la entrada de una compuerta AND común para generar la señal de coincidencia para M_i . M_i será un 1 lógico si ocurre una coincidencia y 0 si no ocurre. Nótese que si el registro de clave contiene sólo ceros, la salida M_i será un 1, sin considerar el valor de A o de la palabra. Debe evitarse que esto ocurra durante la operación normal.

Operación de lectura

Si más de una palabra de memoria coincide con el campo de argumento sin enmascarar, todas las palabras que coinciden tendrán 1 en la posición correspondiente del registro de coincidencia. (Es necesario revisar los bits del registro de coincidencia, uno a la vez.) Las que coinciden se leen en secuencia al aplicar una señal de lectura a cada línea de palabra cuyo bit M_i es un 1.

En la mayoría de las aplicaciones, la memoria asociativa almacena una tabla sin dos datos idénticos bajo una cierta clave. En este caso, sólo una palabra puede coincidir con el campo de argumento sin enmascarar. Al conectar la salida M_i directamente a la línea de lectura, en la misma posición de palabra (en lugar del registro M), el contenido de la palabra que coincide se presentará en forma automática en las líneas de salida y no será necesaria una señal de comando de lectura especial. Además, si excluimos las palabras que contienen cero, una salida de todos cero indicará que no ocurrió una coincidencia y que el dato buscado no está disponible en memoria.

Operación de escritura

Una memoria asociativa debe contar con la posibilidad de escribir para almacenar la información que se va a buscar. Escribir en una memoria asociativa puede tomar diferentes formas, dependiendo de la aplicación. Si se carga la memoria con nueva información una sola vez, antes de una operación de búsqueda, después puede escribirse al direccionar cada localidad en secuencia. Esto hará del dispositivo una memoria de acceso aleatorio para escribir y una memoria de contenido direccionable para leer. La ventaja aquí, es que puede decodificarse la dirección para entrada como en una memoria de acceso aleatorio. Por lo tanto, en lugar de tener m líneas de dirección, una para cada palabra en la memoria, puede reducirse la cantidad de líneas de dirección mediante el decodificador a d líneas, en donde $m = 2^d$.

Si tienen que borrarse palabras no deseadas e insertarse nuevas palabras una a la vez, se necesita un registro especial para diferenciar las palabras activas e inactivas. Este registro, en algunas ocasiones llamado *registro de identificación*, tendrá tantos bits como palabras en la memoria. Para cada palabra activa almacenada en la memoria, el bit correspondiente en el registro de identificación se activa en 1. Se borra una palabra de la memoria al borrar su bit de identificación a 0. Las palabras se almacenan en la memoria al revisar el registro de etiqueta hasta que se encuentra el primer bit 0. Esto proporciona la primera palabra inactiva disponible y una posición

relativamente distantes en el rango de dirección (múltiplos de las 512 posiciones en este ejemplo).

Para ver cómo opera la organización de mapeo directo, consideremos el ejemplo numérico que se muestra en la figura 12-13. La palabra en la dirección cero está almacenada en ese momento en caché (índice = 000, identificación = 00, datos = 1220). Supongamos ahora que la CPU desea accesar la palabra en la dirección 02000. La dirección de índice es 000, por lo que se usa para accesar el caché. Se comparan las dos identificaciones. La etiqueta de caché es 00 pero la etiqueta de dirección es 02, lo que no produce una coincidencia. Por lo tanto, se accesa la memoria principal y la palabra de datos 5670 se transfiere a la CPU. La palabra de caché en la dirección de índice 000 se sustituye después con una identificación de 02 y datos de 5670.

El ejemplo de mapeo directo que se acaba de describir utiliza un tamaño de bloque de una palabra. En la figura 12-14 se muestra la misma organización, pero utilizando un tamaño de bloque de ocho palabras. El campo de índice se divide ahora en dos partes. El campo de bloque y el campo de palabra. En un caché de 512 palabras hay 64 bloques de ocho palabras cada uno, porque $64 \times 8 = 512$. El número de bloques se especifica con un campo de 6 bits y la palabra dentro del bloque se especifica con un campo de 3 bits. El campo de identificación almacenado dentro del caché es común a las ocho palabras del mismo bloque. Cada vez que ocurre una falla, debe transferirse un bloque completo de ocho palabras de la memoria principal a la memoria caché. Aunque esto requiere tiempo extra, es muy probable que la razón de acierto mejore con un bloque de mayor tamaño, debido a la naturaleza secuencial de los programas de computadora.

Figura 12-14 Caché de mapeo directo con tamaño de bloque de 8 palabras.

Índice	Identifi-cación	Datos			
Bloque 0	0 1	3 4 5 0	6	6	3
Bloque 1	0 1	6 5 7 8	Identifi-cación	Bloque	Palabra
			Índice		
Bloque 63	0 2				
	0 2	6 7 1 0			

Después, se compara el campo de identificación de la dirección de la CPU con ambas identificaciones en la caché para determinar si ocurre una coincidencia. La comparación lógica se hace mediante una búsqueda asociativa de las etiquetas en el conjunto, similar a una búsqueda de memoria asociativa: de ahí el nombre de *conjunto asociativo*. La razón de acierto mejorará conforme aumenta el tamaño de conjunto porque pueden residir en el caché más palabras con el mismo índice pero con etiquetas diferentes. Sin embargo, un aumento en el tamaño del conjunto aumenta la cantidad de bits en las palabras de caché y requiere una lógica de comparación más compleja.

Cuando ocurre una falla en una caché de conjunto asociativo y el conjunto está completo, es necesario sustituir uno de los conjuntos identificación-datos con un nuevo valor. Los algoritmos de sustitución que se usan con mayor frecuencia son: sustitución aleatoria, primero en entrar, primero en salir (FIFO) y el uso reciente mínimo (*least recently used*, LRU). Con la política de sustitución aleatoria, el control elige un conjunto identificación-dato para sustituirlo al azar. El procedimiento FIFO selecciona para sustitución el conjunto que ha estado más tiempo en el conjunto. El algoritmo LRU selecciona para sustitución el conjunto que la CPU usó hace más tiempo. Tanto FIFO como LRU pueden implantarse al sumar unos cuantos bits extra en cada palabra de caché.

Escritura en la caché

Un aspecto importante de la organización de caché se relaciona con las solicitudes de escritura a memoria. Cuando la CPU encuentra una palabra durante una operación de lectura, la memoria principal no participa en la transferencia. Sin embargo, si la operación es de escritura, hay dos maneras en las que puede proceder el sistema.

El procedimiento más sencillo y de uso más común es actualizar la memoria principal con cada operación de escritura en memoria, y la memoria caché se actualiza en paralelo si contiene la palabra en la dirección especificada. Esto se denomina el método de *escritura simultánea*. Este método tiene la ventaja de que la memoria principal siempre contiene los mismos datos que el caché. Esta característica es importante en sistemas con transferencias de acceso directo a memoria. Asegura que los datos que reciben en la memoria principal son válidos en todas las ocasiones en que un dispositivo de E/S que se comunica con DMA reciba los datos actualizados de manera más reciente.

El segundo procedimiento se denomina método de *escritura al retorno*. En este método sólo se actualiza la operación de caché mediante una operación de escritura. La localidad se marca después mediante una bandera para que, más tarde, cuando la palabra se quite de caché se copie a la memoria principal. La razón de este método de escritura al retorno es que durante el tiempo que una palabra reside en el caché, puede actualizarse varias veces. Sin embargo, mientras la palabra permanezca en el caché, no importa que

algoritmos de sustitución

escritura simultánea

escritura al retorno

páginas y bloques

Mapeo de dirección usando páginas

La implantación de tabla del mapeo de direcciones se simplifica si la información en el espacio de dirección y en el espacio de memoria se divide cada una en grupos de tamaño fijo. La memoria física se separa en grupos de igual tamaño llamados *bloques*, que pueden variar de 64 a 4096 palabras cada uno. El término *página* se refiere a grupos de espacios de dirección del mismo tamaño. Por ejemplo, si una página o bloque consta de 1K palabras, entonces, usando el ejemplo anterior, el espacio de división se divide en 1024 páginas y la memoria principal en 32 bloques. Aunque una página y un bloque se dividen en grupos de 1K palabras, una página hace referencia a la organización de espacio de direccionamiento, mientras que un bloque hace referencia a la organización de espacio de memoria. Los programas también se consideran divididos en dos páginas. Se mueven partes del programa de la memoria auxiliar a la memoria principal en registros iguales al tamaño de una página. En ocasiones se utiliza el término "cuadro de página" para denotar un bloque.

Consideremos una computadora con un espacio de dirección de 8K y un espacio de memoria de 4K. Si dividimos cada uno en grupos de 1K palabras, obtenemos ocho páginas y cuatro bloques, como se muestra en la figura 12-18. En cualquier momento dado, pueden residir en la memoria principal hasta cuatro páginas de espacio de direccionamiento en cualquiera de los cuatro bloques.

El mapeo del espacio de direccionamiento al espacio de memoria se facilita si cada dirección virtual se considera representada mediante dos números: una dirección de número de página y una línea dentro de la página. En una computadora con 2^p palabras por página, los bits p se usan para especificar una dirección de línea y los bits de orden superior restantes

Figura 12-18 Separación de espacios de dirección y de memoria en grupos de 1K palabras.

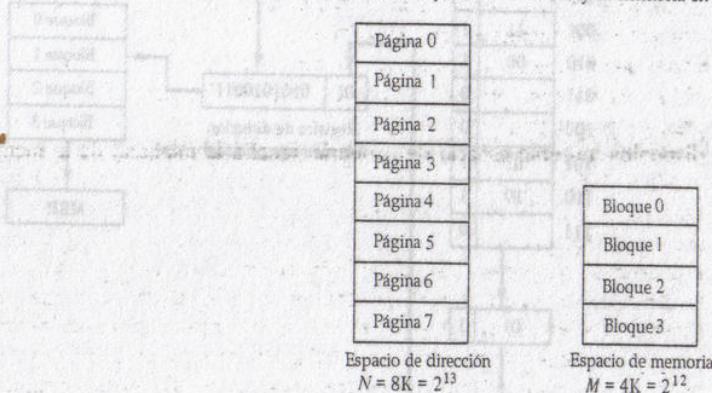


tabla de página de memoria. El contenido de la palabra en la tabla de página de memoria en la misma dirección de número de página se lee al registro intermedio (buffer) de la tabla de memoria. Si el bit de presencia es un 1, entonces el número de bloque se transfiere a los 2 bits de orden superior del registro de dirección de la memoria principal. El número de línea de la dirección virtual se transfiere dentro de los bits de orden inferior del registro de dirección de memoria. Una señal de lectura a la memoria principal transfiere el contenido de la palabra al registro intermedio (buffer) de la memoria principal preparado para que lo use la CPU. Si el bit de presencia en la palabra leída de la tabla de página es 0, esto significa que el contenido de la palabra a que hace referencia la dirección virtual no reside en la memoria principal. Entonces se genera una solicitud al sistema operativo para recuperar la página requerida de la memoria auxiliar y colocarla en la memoria principal antes de reanudar la computación.

Tabla de página de memoria asociativa

Una tabla de página de memoria de acceso aleatorio no es eficiente en relación al uso de almacenamiento. En el ejemplo de la figura 12-19, observamos que se necesitan ocho palabras de memoria, una para cada página, pero al menos cuatro palabras siempre se marcarán vacías porque la memoria principal no puede alojar más de cuatro bloques. En general, un sistema con n páginas y m bloques requeriría una tabla de páginas de memoria de n localidades de las cuales hasta m bloques estarían marcadas con números de bloque y todas las demás estarían vacías. Como un segundo ejemplo numérico, consideremos un espacio de direccionamiento de 1024K palabras y un espacio de memoria de 32K palabras. Si cada página o bloque contiene 1K palabras, la cantidad de páginas es 1024 y la cantidad de bloques 32. La capacidad de la tabla de páginas de memoria debe ser de 1024 palabras y sólo 32 localidades pueden tener un bit de presencia igual a 1. En cualquier momento dado, al menos 992 posiciones estarán vacías y sin usar.

Una manera más eficiente de organizar la tabla de páginas sería construirla con una cantidad de palabras igual a la cantidad de bloques en la memoria principal. De esta manera, se reduce el tamaño de la memoria y cada posición se utiliza por completo. Este método puede implantarse mediante una memoria asociativa en donde cada palabra en la memoria contenga un número de página junto con su número de bloque correspondiente. El campo de página en cada palabra se compara con el número de página en la memoria virtual. Si ocurre una coincidencia, se lee la palabra de la memoria y se extrae su número de bloque correspondiente.

Volvamos a considerar el caso de ocho páginas y cuatro bloques como en el ejemplo de la figura 12-19. Sustituimos la tabla de página de memoria de acceso aleatorio con una memoria asociativa de cuatro palabras como la que se muestra en la figura 12-20. Cada entrada en el conjunto de memoria asociativa consiste en dos campos. Los primeros tres bits especifican un

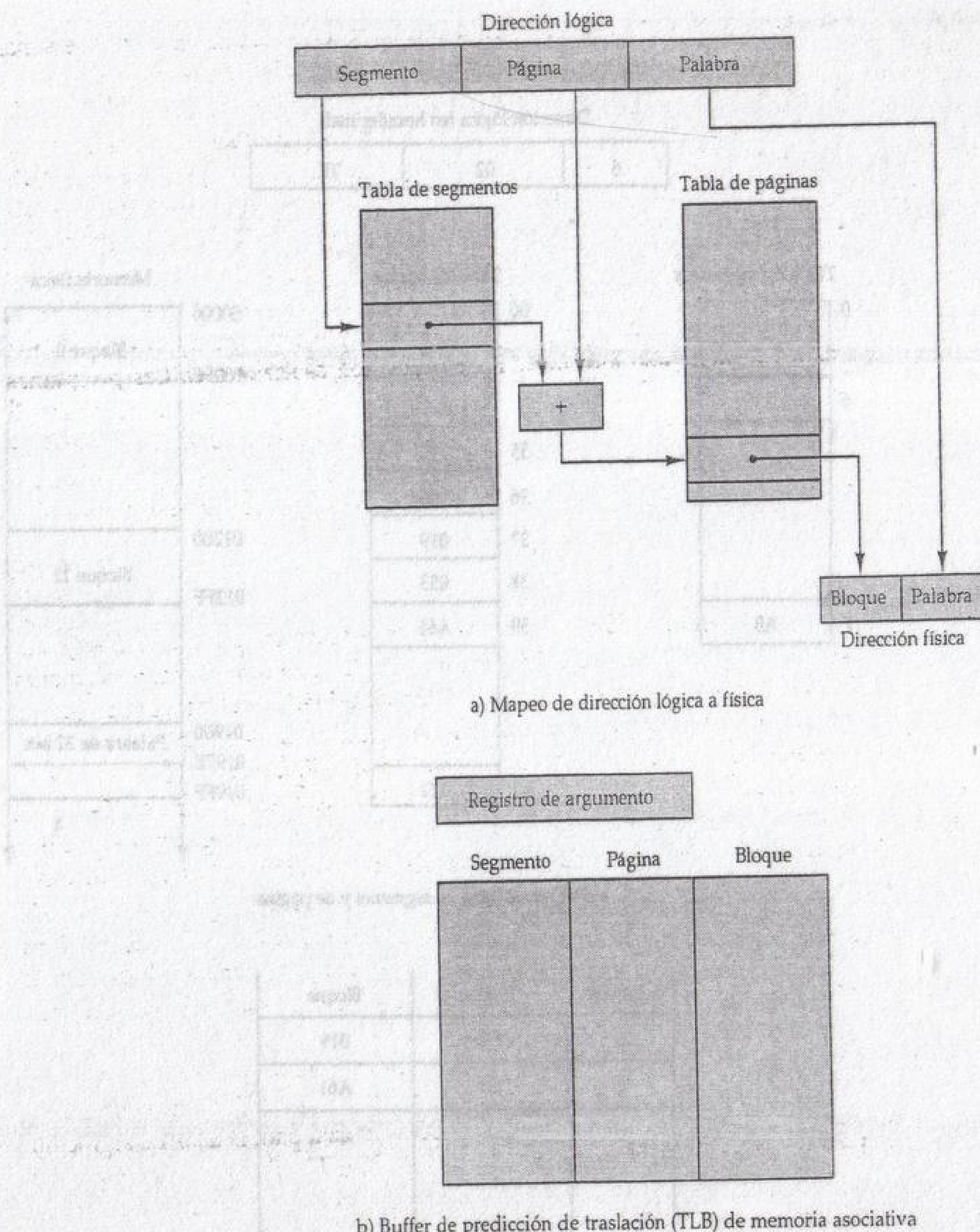


Figura 12-21 Mapeo en una unidad de administración de memoria de página segmentada.

principal. Esto podría hacer lento el sistema de manera muy significativa cuando se compara con un sistema convencional que sólo requiere una referencia a memoria. Para evitar esta insuficiencia en la velocidad, se utiliza una memoria rápida asociativa para contener las entradas de tabla a las que

se ha hecho referencia en forma más reciente. Este tipo de memoria se denomina en ocasiones *buffer de predicción de traslación* (translation lookaside buffer, TLB). La primera vez que se hace referencia a cierto bloque, se introduce su valor junto con el segmento y número de páginas correspondientes dentro de la memoria asociativa, como se muestra en la figura 12-21(b). Por lo tanto, primero se intenta el proceso de mapeo mediante búsqueda asociativa con el segmento y el número de páginas dados. Si tiene éxito, el retraso de mapeo es sólo el de la memoria asociativa. Si no ocurre coincidencia, el mapeo de la tabla más lenta de la figura 12-21(a) se utiliza y el resultado se transforma en la memoria asociativa para referencias futuras.

Ejemplo numérico

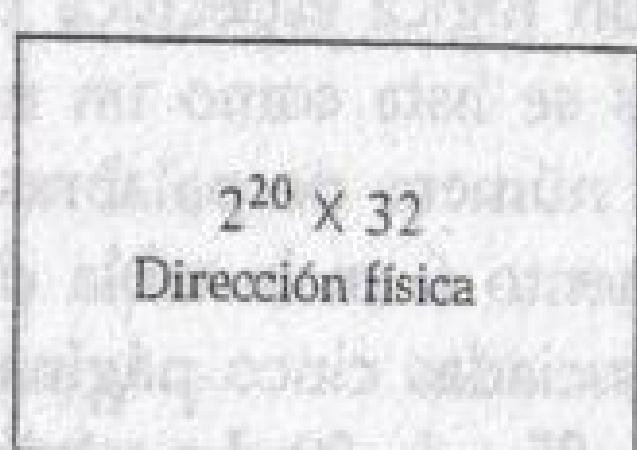
Un ejemplo numérico puede aclarar la operación de la unidad de administración de memoria. Consideremos la dirección lógica de 20 bits especificada en la figura 12-22(a). El número de segmento de 4 bits especifica uno de 16 segmentos posibles. El número de página de 8 bits puede especificar hasta 256 páginas, y el campo de palabra de 8 bits implica un tamaño de página de 256 palabras. Esta configuración permite que cada segmento tenga cualquier cantidad de páginas hasta 256. El segmento más pequeño posible tendrá una o 256 palabras. El segmento más grande posible tendrá 256 páginas, para un total de $256 \times 256 = 64K$ palabras.

La memoria física que se muestra en la figura 12-22(b) consta de 220 palabras de 32 bits cada una. La dirección de 20 bits se divide en dos

Figura 12-22 Un ejemplo de direcciones lógicas y física.

4	8	8
Segmento	Página	Palabra

- a) Formato de dirección lógica: 16 segmentos de 256 páginas cada uno, cada página tiene 256 palabras



12	8
Bloque	Palabra

- b) Formato de dirección física: 4096 bloques de 256 palabras cada uno, cada palabra tiene 32 bits

- Determine las cuatro páginas que residen en la memoria principal después de cada cambio de referencia de página si el algoritmo de sustitución utilizado es: a) FIFO; b) LRU.
- 12-22. Determine las dos direcciones lógicas a partir de la figura 12-24(a) que accesarán la memoria física en la dirección hexadecimal 012AF.
 - 12-23. El espacio de dirección lógica en un sistema de computadora consta de 128 segmentos. Cada segmento puede tener hasta 32 páginas de 4K palabras en cada uno. La memoria física consta de 4K bloques de 4K palabras en cada uno. Formule los formatos de dirección lógica y física.
 - 12-24. Proporcione el número binario de la dirección lógica formulada en el problema 12-23 para el segmento 36 y el número de palabra 2000 en la página 15.

REFERENCIAS

1. Baer, J. L., *Computer Systems Architecture*. Potomac, MD: Computer Science Press, 1980.
2. Dasgupta, S., *Computer Architecture: A Modern Synthesis*, Vol. 1 Nueva York: John Wiley, 1989.
3. Gibson, G. A., *Computer Systems Concepts and Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.
4. Hamacher, V. C., Z. G. Vranesic y S. G. Zaky, *Computer Organization*, 3a. ed. Nueva York: McGraw-Hill, 1990.
5. Hwang, K., y F. A. Briggs, *Computer Architecture and Parallel Processing*. Nueva York: McGraw-Hill, 1984.
6. Kain, R., *Computer Architecture: Software and Hardware*, Vol. 1. Englewood Cliffs, NJ: Prentice Hall, 1989.
7. Langholz, G., J. Francioni, y A. Kandel, *Elements of Computer Organization*. Englewood Cliffs, NJ: Prencite Hall, 1989.
8. Murray, W. D., *Computer and Digital System Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1990.
9. Patterson, D. A., y J. L. Hennessy, *Computer Architecture: A Quantitative Approach*. San Mateo, CA: Morgan Kaufmann Publishers, 1990.
10. Pollard, L. H., *Computer Design and Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1990.
11. Stone, H. S. (ed.), *Introduction to Computer Architecture*, 2a. ed. Chicago: Science Research Associates, 1980.

EN ESTE CAPÍTULO

- 13-1 Características de multiprocesadores
- 13-2 Estructuras de interconexión
- 13-3 Arbitraje entre procesadores
- 13-4 Comunicación y sincronización entre procesadores
- 13-5 Coherencia de caché

13-1 Características de multiprocesadores

Un sistema multiprocesador es una interconexión de dos o más CPU con equipo de memoria y entrada-salida. El término "procesador", en multiprocesador, puede significar una unidad de procesamiento central (CPU) o un procesador de entrada-salida (IOP). Sin embargo, un sistema con una sola CPU y uno o más IOP por lo general no se incluye en la definición de un sistema multiprocesador, a menos que los IOP tengan opciones computacionales comparables a una CPU. Como se define con mayor frecuencia, un sistema multiprocesador implica la existencia de múltiples CPU, aunque por lo general también habrá uno o más IOP. Como se mencionó en la sección 9-1, los multiprocesadores se clasifican como sistemas de flujo de instrucciones múltiple, flujo de datos múltiple (*multiple instruction stream, multiple data stream, MIMD*).

MIDM

Existen varias semejanzas entre sistemas multiprocesadores y multicomputadoras, porque ambos soportan operaciones concurrentes. Sin embargo, existen una importante diferencia entre un sistema con computadoras múltiples y un sistema con procesadores múltiples. Las computadoras se interconectan unas con otras mediante líneas de comunicación para formar una red de computadoras. La red consiste en varias computadoras autónomas que pueden comunicarse o no una con otras. Un sistema multiprocesador lo controla un sistema operativo que proporciona interacción entre los

3. Comutador de barra de cruz.
4. Red de conmutación de etapas múltiples.
5. Sistema de hipercubo.

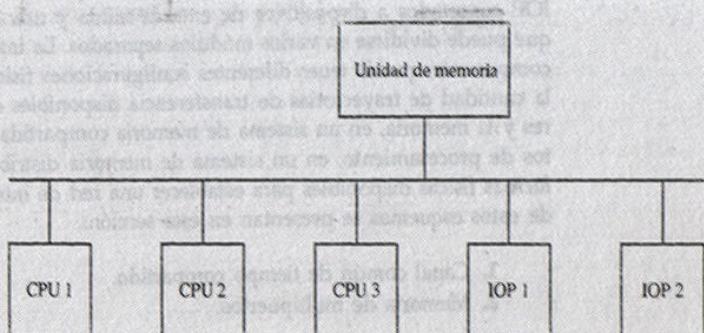
Canal común de tiempo compartido

Un sistema multiprocesador de canal común consta de varios procesadores conectados mediante una trayectoria común a una unidad de memorias. Un canal común de tiempo compartido para cinco procesadores se muestra en la figura 13-1. En cualquier momento dado sólo un procesador puede comunicarse con la memoria o con otro procesador. Las operaciones de transferencia las controla el procesador que controla el canal en ese momento. Cualquier otro procesador que desea iniciar una transferencia debe determinar primero el estado de disponibilidad del canal y, sólo después de que el canal queda disponible, puede el procesador dirigir la unidad destino para iniciar la transferencia.

Se emite un comando para informar a la unidad destino cuál operación se va a ejecutar. La unidad que recibe reconoce su dirección en el canal y responde a las señales de control de la unidad que las envía, después de lo cual se inicia la transferencia. El sistema puede mostrar conflictos de transferencia, dado que todo los procesadores comparten un canal común. Estos conflictos deben resolverse al incorporar un controlador de canal que establezca prioridades entre las unidades solicitantes.

Un sistema de canal común único está limitado a una transferencia a la vez. Esto significa que cuando otro procesador se está comunicando con la memoria, todos los otros procesadores están ocupados con operaciones internas o deben estar inactivos en espera del canal. Como consecuencia, la velocidad de transferencia total dentro del sistema está limitada a la velocidad de la trayectoria única. Con frecuencia, los procesadores en el sistema pueden mantenerse ocupados por medio de la implantación de dos o más

Figura 13-1 Organización de canal común de tiempo compartido.



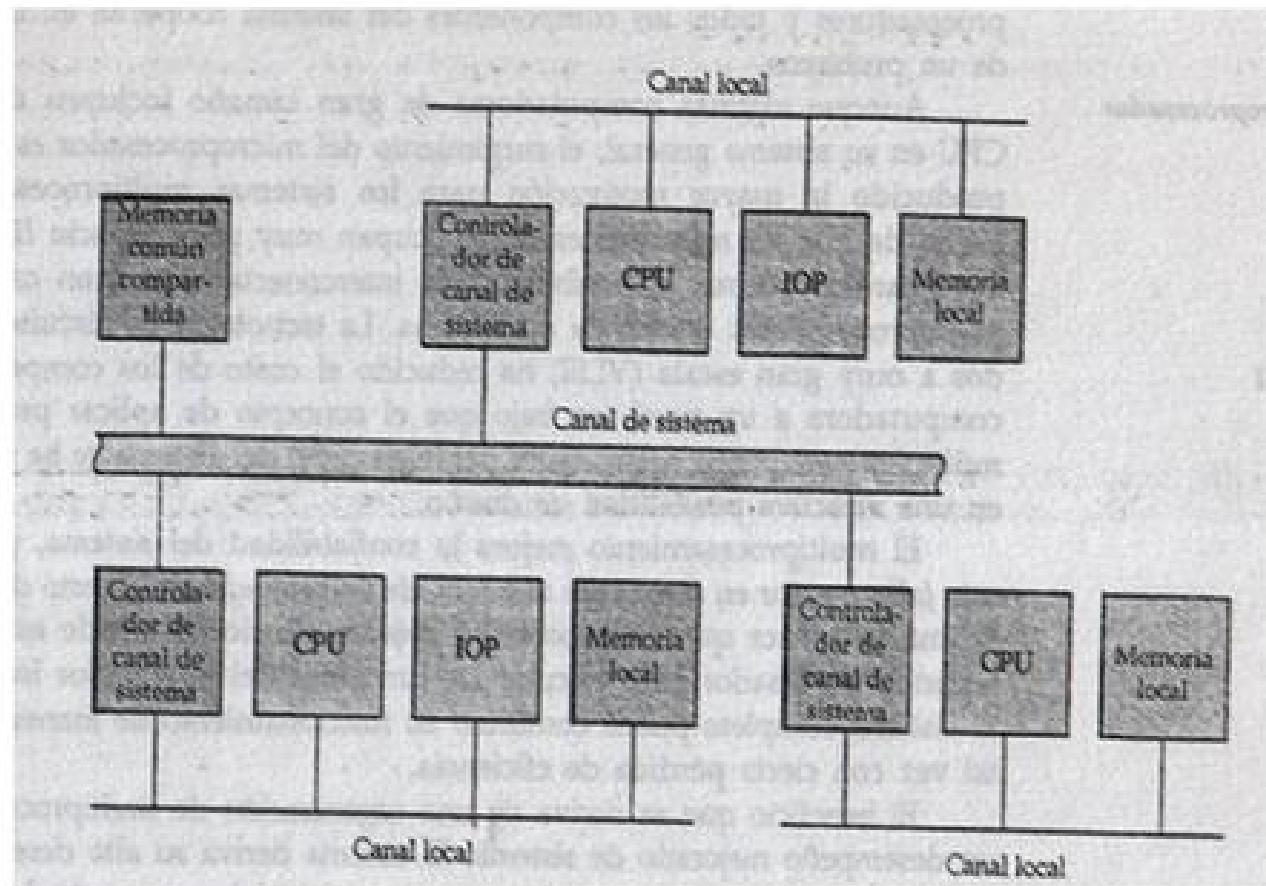


Figura 13-2 Estructura de canal de sistema para multiprocesadores.

canales independientes, para permitir transferencias de canal simultáneas y múltiples. Sin embargo, esto incrementa el costo y la complejidad del sistema.

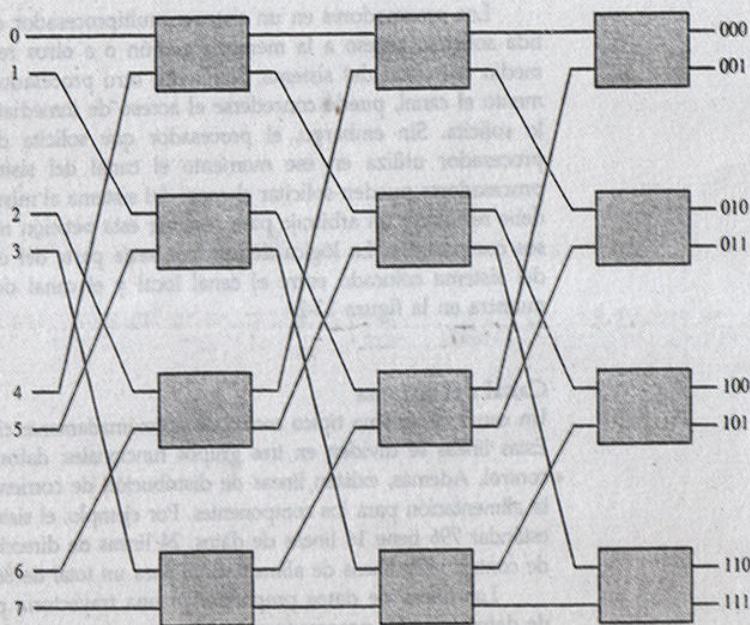
Una implantación más económica de una estructura de doble canal se muestra en la figura 13-2. Aquí tenemos varios canales locales, cada uno conectado a su propia memoria local y a uno o más procesadores. Cada canal local puede estar conectado a una CPU, un IOP o una combinación de procesadores. Un controlador de canal de sistema enlaza cada canal local a un canal de sistema común. Los dispositivos de E/S conectados al IOP local, al igual que a la memoria local, están disponibles para el procesador local. La memoria conectada al canal de sistema común la comparten todos los procesadores. Si un IOP está conectado en forma directa al canal de sistema, los dispositivos de E/S conectados a él, pueden quedar disponibles para todos los procesadores. Sólo un procesador puede comunicarse con la memoria compartida y otros recursos comunes a través del canal de sistema, en cualquier momento dado. Los otros procesadores se mantienen ocupados comunicándose con su memoria local y sus dispositivos de E/S. Parte de la memoria local puede diseñarse como una memoria caché conectada a la CPU (véase la sección 12-6). De esta manera, puede hacerse que el tiempo de acceso promedio de la memoria local se aproxime al tiempo de ciclo de la CPU a la que está conectada.

memoria compartida

salidas. La terminal *B* del conmutador se comporta de manera similar. El conmutador también tiene la capacidad para proporcionar arbitraje entre solicitudes en conflicto. Si las entradas *A* y *B* solicitan la misma terminal de salida, sólo se conectarán una de ellas; la otra se bloqueará.

Al usar el conmutador 2×2 como un bloque de construcción, es posible desarrollar una red de etapas múltiples que controle la comunicación entre varias fuentes y destinos. Para ver cómo se hace esto, consideremos el árbol binario que se muestra en la figura 13-7. Los dos procesadores P_1 y P_2 se conectan mediante conmutadores a ocho módulos de memoria marcados en binario del 000 a 111. La trayectoria de una fuente a un destino la determinan los bits binarios del número destino. El primer bit del número destino determina la salida del conmutador en el primer nivel. El segundo bit especifica la salida del conmutador en el segundo nivel. Y el tercer bit especifica la salida del conmutador en el tercer nivel. Por ejemplo, para conectar P_1 a la memoria 101, es necesario formar una trayectoria de P_1 a la salida 1 en el conmutador de primer nivel, a la salida 0 en el conmutador de segundo nivel y a la salida 1 en el conmutador de tercer nivel. Está claro que P_1 y P_2 pueden conectarse a cualquiera de las ocho memorias. Sin embargo, ciertos patrones de solicitud no pueden satisfacerse en forma simultánea. Por

Figura 13-8 Red de conmutación omega 8×8 .



un código de 2 bits que representa la unidad de prioridad más alta entre las que solicitan el canal. La tabla de verdad del codificador de prioridad puede encontrarse en la tabla 11-2 (véase la sección 11-5). El código de dos bits de la salida del codificador maneja un decodificador 2×4 que habilita la línea de reconocimiento apropiada para conceder el acceso al canal a la unidad de prioridad más alta.

Ahora podemos explicar la función de las señales de arbitraje de canal que se listan en la tabla 13-1. La entrada de prioridad del canal (*bus priority-in*, BPRN) y la salida de prioridad del canal (*bus priority-out*, BPRO) se usan para una conexión en cadena de margaritas de los circuitos de arbitraje de canal. La señal de canal ocupado (*bus busy signal*, BUSY) es una salida de colector abierto que se usa para dar instrucciones a todos los árbitros cuando el canal está ocupado conduciendo una transferencia. La solicitud de canal común (*common bus request*, CBRQ) es también una salida de colector abierto que sirve para dar instrucciones al árbitro si hay otros árbitros de prioridad menor que solicitan el uso del canal del sistema. Las señales utilizadas para construir un procedimiento de arbitraje paralelo son la solicitud de canal (BREQ) y entrada de prioridad del canal (BPRN), que corresponden a las señales de solicitud y reconocimiento de la figura 13-11. Se utiliza el reloj de canal (BCLK) para sincronizar todas las transacciones de canal.

Algoritmos de arbitraje dinámico

Los dos procedimientos de arbitraje de canal que se acaban de describir, utilizan un algoritmo de prioridad estática porque la prioridad de cada dispositivo está fija por la manera en que está conectado al canal. En contraste, un algoritmo de prioridad dinámica proporciona al sistema la posibilidad de cambiar la prioridad de los dispositivos mientras opera el sistema. Ahora discutiremos unos cuantos procedimientos de arbitraje que utilizan algoritmos de prioridad dinámica.

El algoritmo de *cuota de tiempo* asigna una porción de tiempo de longitud fija del tiempo del canal que se ofrece en forma secuencial a cada procesador formando un ciclo que los incluye a todos. El servicio que se proporciona a cada componente del sistema con este esquema, es independiente de su posición a lo largo del canal. No se da preferencia a ningún dispositivo particular porque a cada uno se le asigna la misma cantidad de tiempo para comunicarse con el canal.

En un sistema de canal que utiliza *encuesta*, la señal de cesión de canal se sustituye con un conjunto de líneas llamadas líneas de encuesta que están conectadas a todas las unidades. El controlador de canal utiliza estas líneas para definir una dirección para cada dispositivo conectado al canal. El controlador de canal recorre en secuencia las direcciones en una manera preestablecida. Cuando un procesador que requiere el acceso reconoce su dirección, activa la línea de canal ocupado y después accesa el canal. Después

cuota de tiempo

encuestar

de varios ciclos de canal, el proceso, de encuesta continúa al elegir un procesador diferente. Por lo general, es programable la secuencia de encuesta y, como resultado, puede alterarse la prioridad de selección bajo el control del programa.

LRU El algoritmo *de uso reciente mínimo* (*least recently used*, LRU) proporciona la prioridad más alta al dispositivo que no ha usado el canal durante el intervalo más largo. Las prioridades se ajustan después de varios ciclos de canal, de acuerdo con el algoritmo LRU. Con este procedimiento, no se favorece a ningún procesador sobre otro, porque las prioridades se cambian en forma dinámica para proporcionar a cada dispositivo una oportunidad de accesar el canal.

FIFO En el esquema *primero en entrar, primero en atenderse*, las solicitudes se atienden en el orden recibido. Para implantar este algoritmo, el controlador de canal establece una lista de espera arreglada de acuerdo con el tiempo en el que llegan las solicitudes de canal. Cada procesador debe esperar su turno para usar el canal con base en primero en entrar, primero en salir (FIFO).

cadena de margaritas rotatoria El procedimiento de *cadena de margaritas rotatoria* es una extensión dinámica del algoritmo de cadena de margaritas. En este esquema, no hay un controlador central de canal, y la línea de prioridad está conectada de la salida de prioridad del último dispositivo de regreso a la entrada de prioridad del primer dispositivo en un ciclo cerrado. Esto es similar a las conexiones que se muestran en la figura 13-10, excepto que la salida PO del árbitro 4 está conectada a la entrada PI del árbitro 1. Cualquier dispositivo que tiene acceso al canal sirve como controlador de canal para el siguiente arbitraje. Cada prioridad de árbitro para un cierto ciclo de canal está determinada por su posición a lo largo de la línea de prioridad de canal del árbitro cuyo procesador controla en ese momento el canal. Una vez que un árbitro entrega un canal, tiene la prioridad más baja.

13-4 Comunicación y sincronización entre procesadores

Los diversos procesadores en un sistema multiprocesador deben recibir una opción para comunicarse uno con el otro. Puede establecerse una trayectoria de comunicación mediante canales comunes de entrada-salida. En un sistema de multiprocesador de memoria compartida, el procedimiento más común es tener una parte de la memoria disponible para todos los procesadores. El uso principal de la memoria común es actuar como un centro de mensajes similar a un buzón, en el cual cada procesador puede dejar mensajes para los otros procesadores y recoger los mensajes dirigidos a él.

El procesador que envía estructura una solicitud, un mensaje o un procedimiento y lo coloca en el buzón de memoria. Los bits de estado que residen en la memoria común se utilizan, por lo general, para indicar la condición del buzón, si tiene información significativa y a cual procesador está dirigida. El procesador que recibe puede comprobar de manera periódica

ca el buzón para determinar si hay mensajes válidos para él. El tiempo de respuesta para este procedimiento puede ser muy largo, porque un procesador reconocerá una solicitud sólo cuando hace una encuesta de mensajes. Un procedimiento más eficiente para el procesador que envía es avisar al procesador que recibe en forma directa mediante una señal de interrupción. Esto puede conseguirse por medio de una interrupción entre procesadores iniciada por programa con una instrucción en el programa de un procesador, el cual, cuando se ejecuta, produce una condición de interrupción externa en un segundo procesador. Esto avisa al procesador interrumpido de que el procesador que interrumpe insertó un nuevo mensaje.

Además de la memoria compartida, un sistema multiprocesador puede tener otros recursos compartidos. Por ejemplo, una unidad de almacenamiento de disco magnético conectada a un IOP puede estar disponible para todas las CPU. Esto proporciona una opción para compartir programas de sistema almacenados en el disco. Puede establecerse una trayectoria de comunicación entre dos CPU por medio de un enlace entre dos IOP asociados con dos CPU diferentes. Este tipo de enlace permite que cada CPU trate a la otra como un dispositivo de E/S para que el mensaje pueda transferirse por medio de la trayectoria E/S.

Para evitar un uso conflictivo de los recursos compartidos por varios procesadores debe preverse la asignación de los recursos a los procesadores. Esta tarea se asigna al sistema operativo. Existen tres organizaciones que deben usarse en el diseño de sistemas operativos para multiprocesadores: configuración amo-esclavo, sistema operativo separado y sistema operativo distribuido.

En un modo amo-esclavo, un procesador, denominado el amo, ejecuta siempre las funciones del sistema operativo. Los procesadores restantes, llamados esclavos, no ejecutan funciones del sistema operativo. Si un procesador esclavo necesita un servicio del sistema operativo, debe solicitarlo al interrumpir al maestro y esperar hasta que pueda interrumpirse el programa actual.

En una organización de sistema operativo separado, cada procesador puede ejecutar las rutinas de sistema operativo que necesita. Esta organización es más conveniente para sistemas de memoria distribuida en los cuales cada procesador puede tener su propia copia de todo el sistema operativo.

En la organización de sistema operativo distribuido, las rutinas de sistema operativo se distribuyen mediante los procesadores disponibles. Sin embargo, cada función de sistema operativo particular se asigna a un solo procesador a la vez. Este tipo de organización se denomina también sistema operativo flotante, porque las rutinas flotan de un procesador a otro y la ejecución de las rutinas puede asignarse a diferentes procesadores en diversos momentos.

En un sistema de multiprocesador de memoria distribuida, se distribuye la memoria entre los procesadores y no hay memoria compartida para pasar información. La comunicación entre los procesadores es mediante el

paso de mensajes a través de canales de E/S. La comunicación la inicia un procesador que solicita un procedimiento que reside en la memoria del procesador con el que se desea comunicar. Cuando el procesador que envía y el procesador que recibe se nombran uno al otro como fuente y destino, se establece un canal de comunicación. Después se envía un mensaje con un encabezado y varios objetos de datos usados para comunicarse entre nodos. Puede haber varias trayectorias posibles disponibles para enviar el mensaje entre cualesquiera dos nodos. El sistema operativo en cada nodo contiene información de direccionamiento que indica las trayectorias alternativas que se pueden usar para enviar un mensaje a otros nodos. La eficiencia de comunicación de la red de procesadores depende del protocolo de direccionamiento de comunicación, la velocidad del procesador, la velocidad de enlace de datos y la topología de la red.

Sincronización entre procesadores

El conjunto de instrucciones de un multiprocesador contiene instrucciones básicas que se utilizan para implantar la comunicación y la sincronización entre procesos que cooperan. La comunicación se refiere al intercambio de datos entre procesos diferentes. Por ejemplo, los parámetros que se pasan a un procedimiento en un procesador distinto constituyen una comunicación entre procesadores. La sincronización se refiere al caso especial en el cual los datos que se usan para comunicar entre procesadores son información de control. Se necesita la sincronización para imponer la secuencia de procesos correcta para asegurar el acceso mutuamente exclusivo a datos compartidos que se pueden escribir.

Los sistemas multiprocesador por lo general incluyen varios mecanismos para manejar la sincronización de los recursos. La circuitería implanta en forma directa primitivas de nivel bajo. Estas primitivas son los mecanismos básicos que imponen la exclusión mutua para mecanismos más complejos implantados en programa. Se han desarrollado varios mecanismos de circuitería para la exclusión mutua. Uno de los métodos más populares es el uso de un semáforo binario.

Exclusión mutua con semáforo

Un sistema multiprocesador que funciona en forma adecuada debe proporcionar un mecanismo que garantice el acceso ordenado a la memoria y otros recursos compartidos. Es necesario evitar que dos o más procesadores cambien simultáneamente los datos. Este mecanismo se denomina *exclusión mutua*. Debe proporcionarse exclusión mutua en un sistema multiprocesador para permitir que un procesador excluya o asegure el acceso a un recurso compartido por parte de otros procesadores cuando está en su *sección crítica*. Una sección crítica es una secuencia de programa que, una vez comenzada, debe terminar su ejecución antes que otro procesador accese el mismo recurso compartido.

Una variable binaria llamada *semáforo* se utiliza con frecuencia para indicar si un procesador ejecuta o no una sección crítica. Un semáforo es una bandera controlada por el programa que está almacenada en una localidad de memoria que pueden accesar todos los procesadores. Cuando el semáforo es igual a 1, significa que un procesador ejecuta un programa crítico, por lo que la memoria compartida no está disponibles para los otros procesadores. Cuando el semáforo es igual a 0, la memoria compartida está disponible para cualquier procesador que lo solicite. Los procesadores que comparten el mismo segmento de memoria aceptan no utilizar el segmento de memoria a menos que el semáforo sea igual a 0, lo que indica que esa memoria está disponible. También aceptan activar el semáforo en 1, cuando ejecutan una sección crítica y desactivarlo a 0 cuando terminan.

Probar y activar el semáforo es, por sí misma, una operación crítica y debe efectuarse como una operación única indivisible. Si no, dos o más procesadores pueden probar el semáforo al mismo tiempo y después cada uno activarlo, lo que permitiría que ambos entraran a una sección crítica al mismo tiempo. Esta acción permitiría la ejecución simultánea de secciones críticas, lo que produciría una inicialización errónea de los parámetros de control y la pérdida de alguna información esencial.

mecanismo de seguro Puede inicializarse un semáforo mediante una instrucción de probar y activar junto con un mecanismo de seguro por la circuitería. Un seguro por la circuitería es una señal generada por el procesador que sirve para evitar que otros procesadores utilicen el canal del sistema mientras la señal está activa. La instrucción probar y activar prueba y activa un semáforo y activa el mecanismo de seguro durante el tiempo que se ejecuta la instrucción. Esto evita que otros procesadores cambien el semáforo entre el tiempo que el procesador lo prueba y el tiempo en que lo activa. Consideremos que el semáforo es un bit en la posición menos significativa de una palabra de memoria cuya dirección se representa mediante SEM. Sea el mnemónico TSL el nombre de la "operación probar y activar mientras esté asegurado". La instrucción

TSL SEM

se ejecutará en dos ciclos de memoria (el primero para leer y el segundo para escribir) sin interferencia, de la manera siguiente:

$R \leftarrow M[SEM]$	Probar semáforo
$M[SEM] \leftarrow 1$	Activar semáforo

Se prueba el semáforo al transferir su valor a un registro de procesador R y después se activa en 1. El valor en R determina qué hacer después. Si el procesador encuentra que $R = 1$, sabe que el semáforo estaba activado originalmente, (el hecho de que se vuelva a activar no cambia el valor del semáforo). Esto significa que otro procesador ejecuta otra sección crítica, por lo que el procesador que probó el semáforo no accesa la memoria compar-

tida. Si $R = 0$, eso significa que la memoria común (o el recurso compartido que representa el semáforo) está disponible. El semáforo se activa en 1 para evitar que otros procesadores accesen la memoria. Ahora el procesador puede ejecutar la sección crítica. La última instrucción en el programa debe desactivar la localidad SEM a 0, para entregar el recurso compartido a otros procesadores.

Nótese que la señal de seguro debe estar activa durante la ejecución de la instrucción probar y activar. No tiene que estar activa una vez que se activa el semáforo. Por lo tanto, el mecanismo de seguro evita que otros procesadores accesen la memoria mientras se activa el semáforo. El semáforo mismo, cuando está activado, evita que otros procesadores accesen la memoria compartida mientras ejecuta una sección crítica.

13-5 Coherencia de caché

La operación de memoria caché se explica en la sección 12-6. La ventaja principal del caché es su capacidad para reducir el tiempo de acceso promedio en los uniprocesadores. Cuando el procesador encuentra una palabra en caché durante una operación de lectura, la memoria principal no participa en la transferencia. Si la operación es de escritura, existen dos procedimientos comunes para actualizar la memoria. En la política *escritura simultánea*, se actualizan las memorias caché y principal con cada operación de escritura. En la política *escritura al retorno* sólo se actualiza caché y se marca la localidad para que pueda copiarse después en la memoria principal.

En un sistema multiprocesador de memoria compartida, todos los procesadores comparten una memoria común. Además, cada procesador puede tener una memoria local, parte de la cual o toda puede ser caché. La razón obligatoria de tener caché separadas para cada procesador es reducir el tiempo de acceso promedio en cada uno de ellos. La misma información puede residir en varias copias en algunos caché y en la memoria principal. Para asegurar la capacidad del sistema de ejecutar operaciones de memoria en forma correcta las copias múltiples deben de ser idénticas. Este requisito impone un problema de *coherencia de caché*. Un esquema de memoria es coherente si el valor que se retorna en la instrucción de carga es siempre el valor proporcionado por la última instrucción de almacenamiento con la misma dirección. Sin una solución apropiada al problema de coherencia de caché, puede utilizarse memoria caché en multiprocesadores orientados a canal con dos o más procesadores.

Condiciones para incoherencia

Los problemas de coherencia de caché existen en los multiprocesadores con caché privados por la necesidad de compartir datos que se pueden escribir. Los datos de sólo lectura pueden contestarse de manera segura sin mecanis-

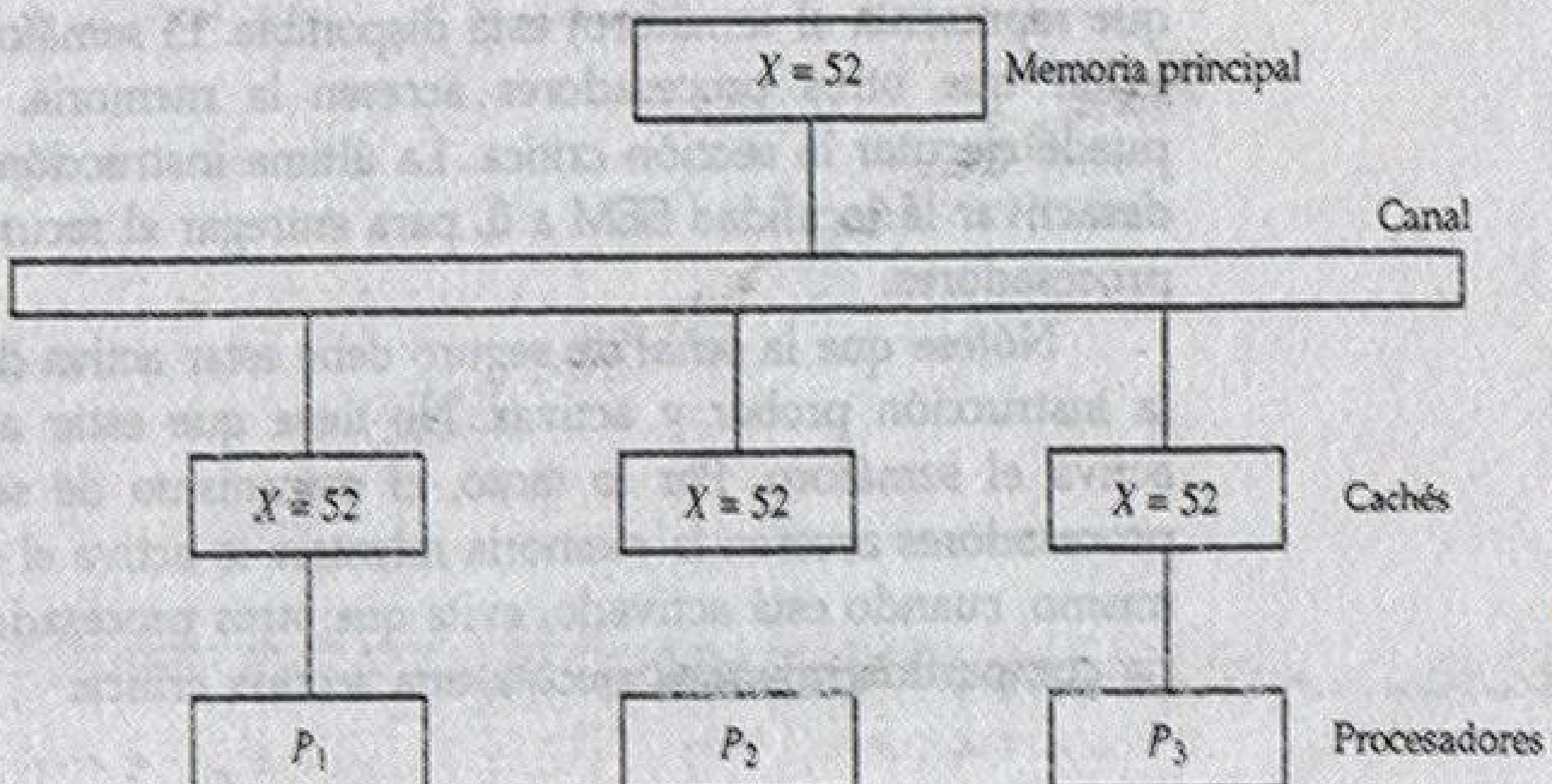
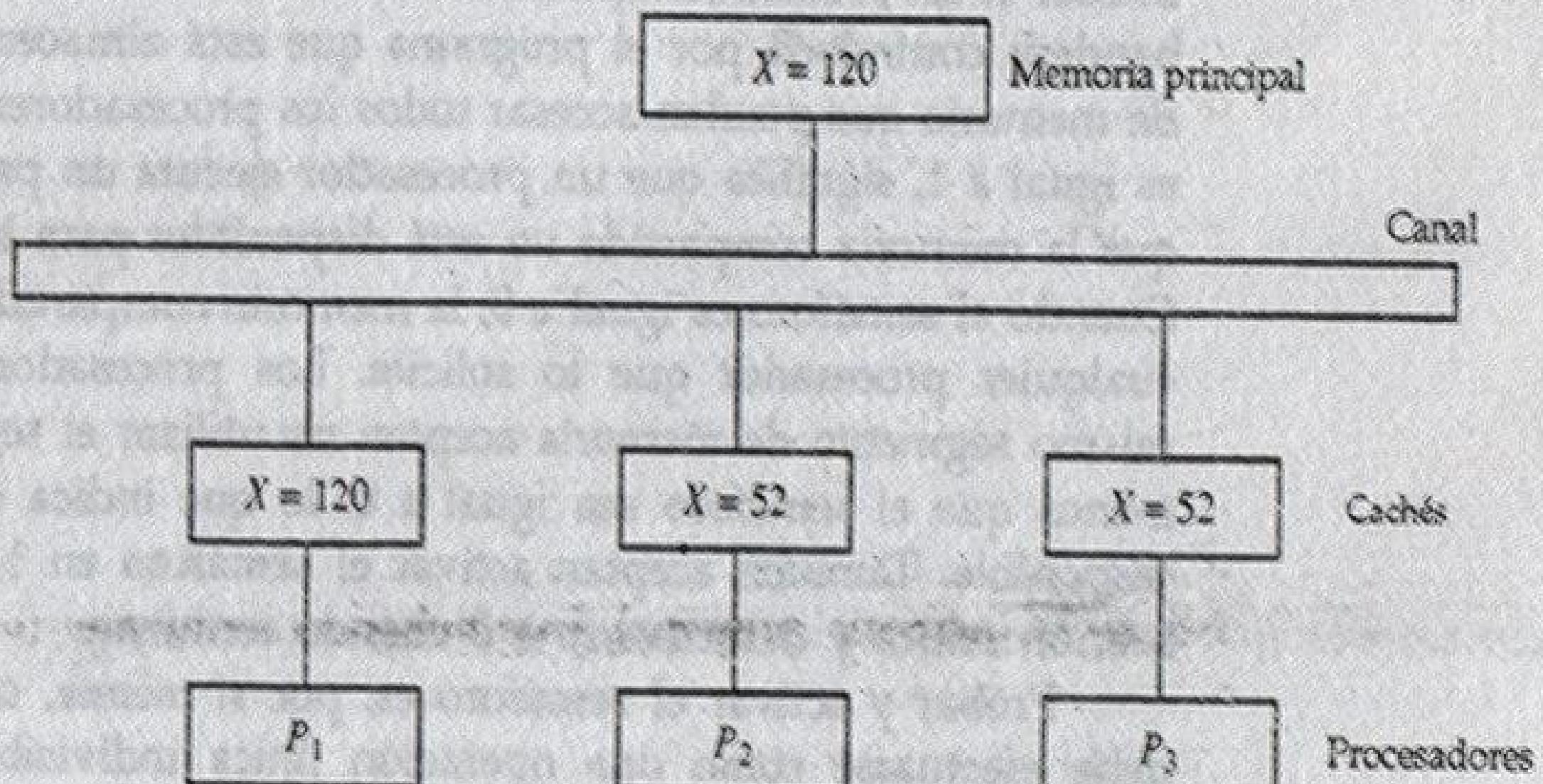


Figura 13-12 Configuración de caché después de una carga X .

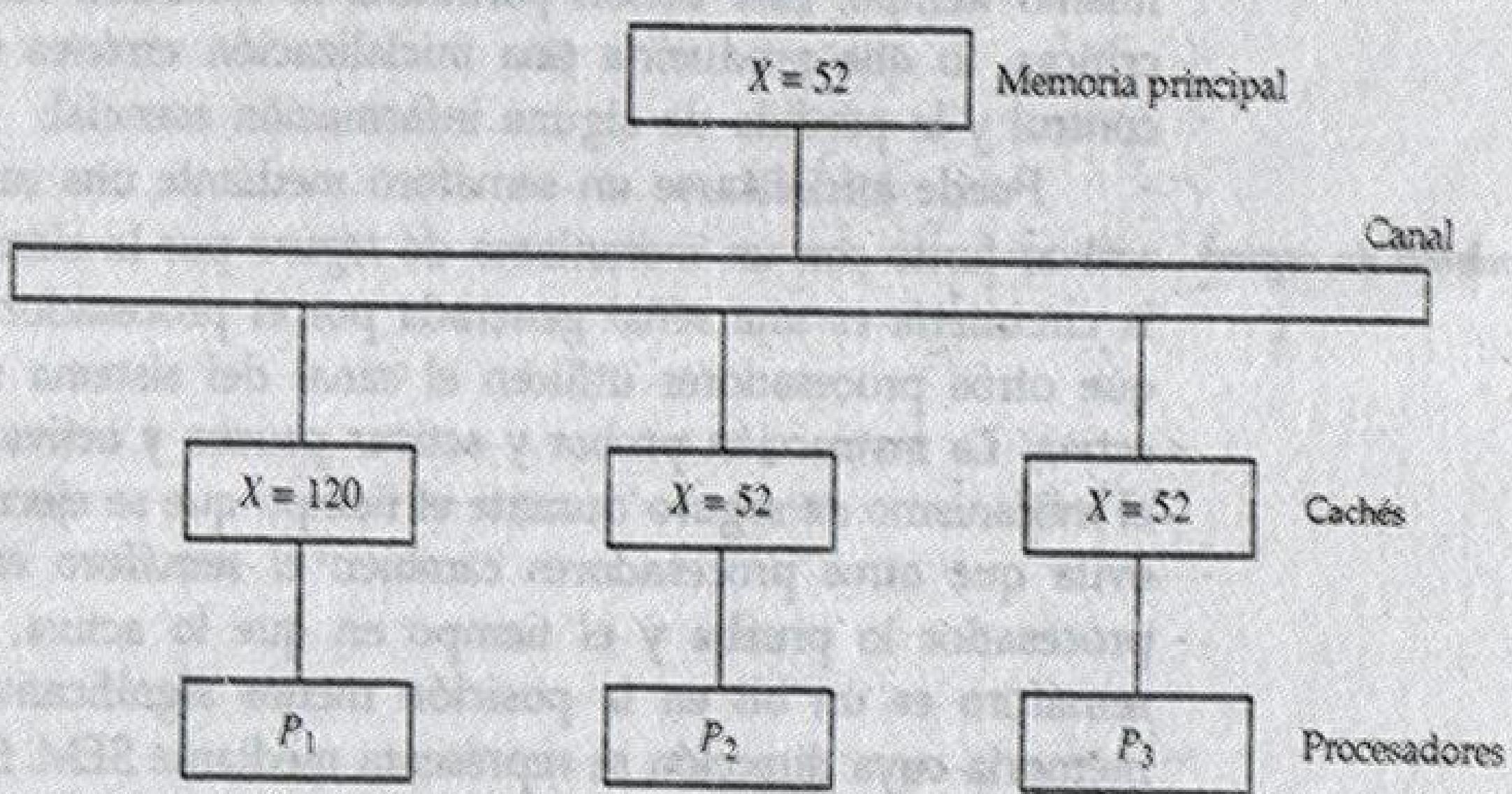
mos de imposición de coherencia de caché. Para mostrar el problema, consideremos la configuración de tres procesadores con caché privadas que se muestra en la figura 13-12. En algún momento durante la ejecución, un elemento X de la memoria principal se carga en los tres procesadores, P_1 , P_2 y P_3 . Como consecuencia, también se copia en las caché privadas de los tres procesadores. Por simplicidad, consideramos que X tiene el valor 52. Al cargar X en los tres procesadores da como resultado copias consistentes en las caché y en la memoria principal.

Si uno de los procesadores ejecuta un almacenamiento a X , las copias de X en las caché se vuelven inconsistentes. Una carga por alguno de los otros procesadores no regresará el valor más reciente. Dependiendo de la política de actualización de memoria utilizada en el caché, la memoria principal puede también ser inconsistente en relación con el caché. Esto se muestra en la figura 13-13. Un almacenamiento a X (del valor 120) dentro del caché del procesador P_1 actualiza la memoria al nuevo valor en una política de n escritura simultánea. Una política de escritura simultánea mantiene la consistencia entre la memoria y el caché de origen, pero los otros dos caché son inconsistentes porque todavía contienen el valor anterior. En una política de escritura al retorno, la memoria principal no se actualiza en el momento del almacenamiento. Las copias en las otras dos caché y en la memoria principal son inconsistentes. Eventualmente, se actualiza la memoria cuando los datos modificados en el caché se copian a ella.

Otra configuración que puede producir problemas de consistencia es una actividad de acceso directo a memoria (DMA) junto con un IOP conectado al canal del sistema. En el caso de entrada, el DMA puede modificar las localidades en la memoria principal que también residen en caché sin actualizar caché. Durante una salida DMA, las localidades de memoria pueden leerse antes de que se actualicen del caché cuando se usa una política de escritura al retorno. Puede superarse la incoherencia de memoria basada



a) con política de caché de escritura simultánea



b) con política de caché de escritura al retorno

Figura 13-13 Configuración de caché después de un almacenamiento X mediante el procesador P_1 .

en E/S, al hacer que el IOP participe en la solución coherente del caché que se adopta en el sistema.

Soluciones al problema de coherencia de caché

Se han propuesto varios esquemas para resolver el problema de coherencia de caché en multiprocesadores de memoria compartida. Aquí analizamos algunos de estos esquemas brevemente. Véase las referencias 3 y 10 para un análisis más detallado.

Un esquema simple es no permitir cachés privados para cada procesador y tener una memoria caché compartida asociada con la memoria principal.

Cada acceso de datos se hace a la caché compartida. Este método no respeta el principio de cercanía de la CPU al caché y aumenta el tiempo de acceso a memoria promedio. En efecto, este esquema resuelve el problema evitándolo.

Por consideraciones de desempeño es deseable conectar una caché privada a cada procesador. Un esquema que se ha usado permite que sólo se almacenen en caché datos no compartidos y de sólo lectura. Tales datos se llaman *apropiados para caché*. Los datos compartidos que se pueden escribir no son adecuados para caché. El compilador debe señalar los datos como adecuados o no para caché y la circuitería del sistema asegura que sólo se almacenen en las caché datos adecuados. Los datos no adecuados permanecen en la memoria principal. Este método restringe el tipo de datos almacenados en caché e introduce una sobrecarga de programación extra que puede degradar el desempeño.

Un esquema que permite que existan datos que se pueden escribir en, al menos, una caché es un método que emplea una tabla global centralizada en su compilador. El estado de los bloques de memoria se almacena en la tabla global central. Cada bloque se identifica como de sólo lectura (RO) o de lectura y escritura (RW). Todas las caché pueden tener copias de los bloques identificados como RO. Sólo una caché puede tener una copia de un bloque RW. Por lo tanto, si se actualizan los datos en caché con un bloque RW, no se afecta a las otras caché porque no tienen una copia de este bloque.

El problema de coherencia de caché puede resolverse mediante una combinación de programación y circuitería o sólo mediante esquemas de circuitería. Los dos métodos mencionados antes, utilizan procedimientos basados en programación que requieren la capacidad de señalar información para deshabilitar el almacenamiento en caché de los datos compartidos que se pueden escribir. Las soluciones sólo de circuitería las maneja automáticamente la circuitería y tienen la ventaja de una mayor velocidad y transparencia del programa. En la solución de circuitería, el controlador de caché está diseñado especialmente para permitirle monitorear todas las solicitudes de canal por parte de la CPU y los IOP. Todas las caché conectadas al canal monitorean constantemente la red en busca de posibles operaciones de escritura. Dependiendo del método usado, deben después actualizar o invalidar sus propias copias de caché cuando se detecta una coincidencia. El controlador de canal que monitorea esta acción se denomina *controlador verificador (snoopy) de caché*. Esta es básicamente una unidad de circuito diseñada para mantener un mecanismo de verificación de canal sobre todas las caché conectadas a él.

Se han propuesto varios esquemas para resolver el problema de coherencia de caché mediante un protocolo verificador de caché. El método más simple es adoptar una política de escritura simultánea y usar el procedimiento siguiente. Todos los controladores verificadores vigilan el canal en busca de operaciones de almacenamiento. Cuando se actualiza una palabra en caché al escribirla, la localidad correspondiente en la memoria principal se

controlador
verificador de caché

actualiza también. Los controladores verificadores locales en todas las otras caché comprueban su memoria para determinar si tienen una copia de la palabra sobre la que se ha escrito. Si existe una copia remota en un caché, se marca la localidad como inválida. Debido a que todas las caché verifican todas las escrituras de canal, cada vez que se escribe una palabra, el efecto neto es actualizarla en la caché original y en la memoria principal y quitarla de las otras caché. Si en algún momento futuro un procesador accesa el dato inválido de su caché, la respuesta es equivalente a una falla de caché, y el dato actualizado se transfiere de la memoria principal. De esta manera, se evitan las versiones inconsistentes.

PROBLEMAS

- 13-1.** Analice la diferencia entre los multiprocesadores con memoria distribuida y los de memoria compartida del punto de vista de organización de circuitería y técnicas de programación.
- 13-2.** ¿Cuál es el propósito del controlador de canal de sistema que se muestra en la figura 13-2? Explique como puede diseñarse el sistema para distinguir entre las referencias a memoria local y las referencias a memoria compartida común.
- 13-3.** ¿Cuántos puntos de conmutador hay en una red de conmutador de barra de cruz que conectan p procesadores a m módulos de memoria?
- 13-4.** La red de conmutación omega 8×8 de la figura 13-8 tiene tres etapas con cuatro conmutadores en cada etapa, para un total de 12 conmutadores. ¿Cuántas etapas y conmutadores por etapa se necesitan en una red de conmutación omega $n \times n$?
- 13-5.** Supongamos que se rompe la línea entre el conmutador en el primer renglón, segunda columna y el conmutador en el segundo renglón, tercera columna en la red de conmutación omega de la figura 13-8. ¿Cuáles trayectorias se desconectarán?
- 13-6.** Construya un diagrama para una red de conmutación omega 4×4 . Muestre las especificaciones de conmutador requeridas para conectar la entrada 3 a la salida 1.
- 13-7.** Se usan tres tipos de conmutadores para diseñar una red de interconexión de etapas múltiples: un conmutador de intercambio con dos entradas y dos salidas como en la figura 13-6, un conmutador de arbitraje con dos entradas y una salida y un conmutador de distribución con una entrada y dos salidas.
 - a. Muestre cómo operan los conmutadores de arbitraje y distribución.
 - b. Utilizando conmutadores de arbitraje e intercambio, construya una red 8×4 con una trayectoria única entre cualquier fuente y cualquier destino.
 - c. Utilizando conmutadores de distribución e intercambio, construya una red 4×8 con una trayectoria única entre cualquier fuente y cualquier destino.

- 13-8. Dibuje un diagrama que muestre la estructura de una red de hipercubo de cuatro dimensiones. Liste todas las trayectorias disponibles del nodo 7 al nodo 9 que utilicen la mínima cantidad de nodos intermedios.
- 13-9. Dibuje un diagrama lógico utilizando compuertas y flip-flops que muestren el circuito de una etapa de árbito de canal en el esquema de arbitraje de cadena de margaritas de la figura 13-10.
- 13-10. Al principio, el canal controlado por la lógica de arbitraje paralelo que se muestra en la figura 13-11 está inactivo. Después, los dispositivos 2 y 3 solicitan el canal al mismo tiempo. Especifique los valores binarios de entrada y salida en el codificador y decodificador y determine cuál árbito de canal se reconoce.
- 13-11. Muestre cómo puede modificarse la lógica de arbitraje de la figura 13-10 para proporcionar un procedimiento de arbitraje de cadena de margaritas circular. Explique cómo se determina la prioridad una vez que se deshabilita la línea de canal.
- 13-12. Considere una topología de canal en la cual dos procesadores se comunican a través de un acoplador en la memoria compartida. Cuando un procesador desea comunicarse con el otro, pone la información en el acoplador de memoria y activa una bandera. Periódicamente, el otro procesador comprueba las banderas para determinar si tiene información por recibir. ¿Qué puede hacerse para asegurar la sincronización apropiada y minimizar el tiempo entre el envío y la recepción de la información?
- 13-13. Describa la siguiente terminología asociada con multiprocesadores: a) exclusión mutua; b) sección crítica; c) seguro de circuitería; d) semáforo; e) instrucción probar y activar.
- 13-14. ¿Qué es coherencia de caché y por qué es importante en sistemas de multiprocesadores de memoria compartida? ¿Cómo puede resolverse el problema con un controlador verificador de caché?

REFERENCIAS

1. Dasgupta, S., *Computer Architecture: A Modern Synthesis*, Vol. 2. Nueva York: John Wiley, 1989.
2. DeCegama, A. L., *Parallel Processing Architecture and VLSI Hardware*. Englewood Cliffs, NJ: Prentice Hall, 1989.
3. Dubois M. C., Scheurich, y F. A. Briggs, "Synchronization, Coherence and Event Ordering in Multiprocessors". *IEEE Computer*, Vol. 21, No. 2 (Febrero 1988), pp 9-21.
4. Gibson, G. A., *Computer Systems Concepts and Design*. Englewood Cliffs, NJ: Prentice Hall, 1991.
5. Gorsline, G. W., *Computer Organization: Hardware/Software*, 2a. ed. Englewood Cliffs, NJ: Prentice Hall, 1986.
6. Hays, J. F., *Computer Architecture Organization*, 2a. ed. Nueva York: McGraw-Hill, 1988.

7. Hwang, K., y F. A. Briggs, *Computer Architecture and Parallel Processing*. Nueva York: McGraw-Hill, 1984.
8. Kain, R., *Computer Architecture: Software and Hardware*, Vol. 2 Englewood Cliffs, NJ: Prentice Hall, 1989.
9. Langholz, G., J. Francioni, y A. Kandel, *Elements of Computer Organization*. Englewood Cliffs, NJ: Prentice Hall, 1989.
10. Stenstrom, P., "A survey of Cache Coherence Schemes for Multiprocessors". *IEEE Computer*, Vol. 23, No. 6 (Junio 1990), pp. 12-24.
11. Stone, H. S., *High-Performance Computer Architecture*. 2a. ed. Reading, MA: Addison-Wesley, 1990.
12. Tabak, D., *Multiprocessors*. Englewood Cliffs, NJ: Prentice Hall, 1990.



ARQUITECTURA DE COMPUTADORAS

TERCERA EDICIÓN

M. Morris Mano

Más de 100 000 estudiantes y profesionales han aprendido los principios del diseño de sistemas digitales y de la arquitectura de computadoras con los textos de M. Morris Mano. La tercera edición de *Arquitectura de Computadoras*, continúa esta tradición de amplia divulgación. La claridad de la presentación, el cuidadoso desarrollo de los principios básicos y la progresión hacia técnicas avanzadas no decepcionarán a los lectores que esperan altos estándares de calidad por parte de Mano.

Con una actualización temática minuciosa, la tercera edición contiene un 85% de material nuevo o revisado, incluso nuevos problemas en la mayor parte de los capítulos. Los primeros siete capítulos abarcan el material necesario para la comprensión de la organización y el diseño de computadoras, al igual que la programación de una computadora modelo, utilizando los componentes básicos. Los seis últimos capítulos presentan la organización y la arquitectura de las unidades funcionales independientes de la computadora digital, con énfasis en los temas avanzados.

Características de la tercera edición:

- Un nuevo capítulo sobre procesamiento de vector y de conducto.
- Un nuevo capítulo sobre Multiprocesadores
- Dos nuevas secciones sobre arquitecturas RISC.
- Problemas nuevos en casi todos los capítulos.

Otros libros de M. Morris Mano editados por Prentice Hall:

Diseño digital, segunda edición (1991)

Ingeniería Computacional (1988)

Lógica digital y diseño de computadoras (1979)

ISBN 968-880-361-8



PEARSON
Educación®

Visítenos en:
www.pearsoneducacion.net